NPS-CS-94-010

NAVAL POSTGRADUATE SCHOOL
Monterey, California

**SOFTWARE REFERENCE:
A VIRTUAL WORLD FOR AN
AUTONOMOUS UNDERWATER VEHICLE**

Donald P. Brutzman

December 1994

Approved for public release; distribution is unlimited.

Prepared for:
Naval Postgraduate School
Monterey California 93943

NAVAL POSTGRADUATE SCHOOL
Monterey California

Rear Admiral T.A. Mercer
Superintendent

Harrison Shull
Provost

This report was prepared for the Naval Postgraduate School.

Reproduction of all or part of this report is authorized.

This report was prepared by:

DONALD P. BRUTZMAN

Reviewed by:

MICHAEL J. ZYDA
Professor of Computer Science

TED LEWIS, Chairman
Department of Computer Science

Released by:

PAUL J. MARTO
Dean of Research

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704

1. AGENCY USE ONLY	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Technical

4. TITLE AND SUBTITLE

SOFTWARE REFERENCE:
A VIRTUAL WORLD FOR AN
AUTONOMOUS UNDERWATER VEHICLE

5. FUNDING NUMBERS

6. AUTHOR Donald P. Brutzman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT. This Software Reference documents and summarizes all source code produced for a Ph.D. dissertation constructing an underwater virtual world for an Autonomous Underwater Vehicle (AUV).

A critical bottleneck exists in Autonomous Underwater Vehicle design and development. It is tremendously difficult to observe, communicate with and test underwater robots, because they operate in a remote and hazardous environment where physical dynamics and sensing modalities are counterintuitive.

An underwater virtual world can comprehensively model all salient functional characteristics of the real world in real time. This virtual world is designed from the perspective of the robot, enabling realistic AUV evaluation and testing in the laboratory. Three-dimensional real-time computer graphics are our window into that virtual world.

Visualization of robot interactions within a virtual world permits sophisticated analyses of robot performance that are otherwise unavailable. Sonar visualization permits researchers to accurately "look over the robot's shoulder" or even "see through the robot's eyes" to intuitively understand sensor-environment interactions. Extending the theoretical derivation of a set of six-degree-of-freedom hydrodynamics equations has provided a fully general physics-based model capable of producing highly non-linear yet experimentally-verifiable response in real time.

Distribution of underwater virtual world components enables scalability and real-time response. The IEEE Distributed Interactive Simulation (DIS) protocol is used for compatible live interaction with other virtual worlds. Network connections allow remote access, demonstrated via Multicast Backbone (MBone) audio and video collaboration with researchers at remote locations. Integrating the World-Wide Web allows rapid access to resources distributed across the Internet.

14. SUBJECT TERMS Virtual worlds, autonomous underwater vehicles, robotics, computer graphics, networking, hydrodynamics, real time, artificial intelligence, control systems, sonar, scientific visualization.

15. NUMBER OF PAGES 331

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

ABSTRACT

This Software Reference documents and summarizes all source code produced for a Ph.D. dissertation constructing an underwater virtual world for an Autonomous Underwater Vehicle (AUV).

A critical bottleneck exists in Autonomous Underwater Vehicle (AUV) design and development. It is tremendously difficult to observe, communicate with and test underwater robots, because they operate in a remote and hazardous environment where physical dynamics and sensing modalities are counterintuitive.

An underwater virtual world can comprehensively model all salient functional characteristics of the real world in real time. This virtual world is designed from the perspective of the robot, enabling realistic AUV evaluation and testing in the laboratory. Three-dimensional real-time computer graphics are our window into that virtual world.

Visualization of robot interactions within a virtual world permits sophisticated analyses of robot performance that are otherwise unavailable. Sonar visualization permits researchers to accurately "look over the robot's shoulder" or even "see through the robot's eyes" to intuitively understand sensor-environment interactions. Extending the theoretical derivation of a set of six-degree-of-freedom hydrodynamics equations has provided a fully general physics-based model capable of producing highly non-linear yet experimentally-verifiable response in real time.

Distribution of underwater virtual world components enables scalability and real-time response. The IEEE Distributed Interactive Simulation (DIS) protocol is used for compatible live interaction with other virtual worlds. Network connections allow remote access, demonstrated via Multicast Backbone (MBone) audio and video collaboration with researchers at remote locations. Integrating the World-Wide Web allows rapid access to resources distributed across the Internet.

This dissertation presents the frontier of 3D real-time graphics to support underwater robotics, scientific ocean exploration, sonar visualization and worldwide collaboration.

ACKNOWLEDGEMENTS

Many people helped in this work. Mike Zyda is the best dissertation advisor anyone might hope for. His insight, support and enthusiasm are boundless. Bob McGhee and Tony Healey showed unlimited patience and insight as we explored the frontiers of dynamics modeling. Mike Bailey taught me analytical and discrete event simulation. He and Man-Tak Shing also gave valuable advice on the Ph.D. process. Mike Macedonia's unparalleled understanding of computer networks helped make an entire field intelligible.

Dave Pratt blazed the trail with NPSNET, still the best virtual world around and still improving faster than all the others. Dave provided crucial academic advice and also the financial support which made the *SIGGRAPH 94* exhibit at *The Edge* possible. I am indebted to everyone who helped make that weeklong demonstration possible, especially Shirley Isakari Pratt, John Roesli, Frank Tipton, Jim Vaglia, Matt Johnson, Chris Stapleton, Garry Paxinos, Jackie Ford Morie, Theresa-Marie Rhyne, Paul Barham, John Locke, Steve Zeswitz, Rosalie Johnson, Russ and Sue Whalen, Walt Landaker, Dave Marco, Mike Williams, Terry Williams, Hank Hankins and Hollis Berry. I also thank Peter Purdue, Gordon Bradley, Jim Eagle, Ted Lewis, Mike McCann, Bruce Gritton, Mike Lee, David Warren, Dave Norman, John Sanders, Dick Blidberg, SeHung Kwak, Ron Byrnes, Drew Bennett, Jim Bales, Jim Bellingham, Alan Beam, Claude Brancart, Rodney Luck, John Gambrino, and Larry Ziomek for their help. Support for this research was provided in part by the National Science Foundation under Grant BCS-9306252 to the Naval Postgraduate School.

This work is dedicated with love and thanks to my wife Terri and our children Hilary, Rebecca, Sarah and Patrick.

SOFTWARE REFERENCE

A Virtual World for an Autonomous Underwater Vehicle

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The purpose of this guide is to document the source code developed in the creation of an underwater virtual world for an autonomous underwater vehicle (AUV). It is companion document to a dissertation (Brutzman 94). There are a number of parts to this work. First a users guide is presented which describes how to obtain and install the software described here. Key mission output files then show results produced by the autonomous underwater robot interacting with the virtual world. Remaining chapters present the source code developed for robot execution software, 3D computer graphics, six degree-of-freedom hydrodynamics, a geometric sonar model, networking, and hypermedia using the World-Wide Web and the Multicast Backbone (MBone).

Several languages are included in this source code. Robot execution programs and supporting configuration files are written in the Kernigan and Richie variant of the ***C*** language, and run identically under Irix 5.2 or OS-9 operating systems. Other source programs are written primarily in C++. Source code file name conventions use a *.c* extension for ***C*** language files, and a *.C* extension for ***C++*** language files. Remaining programs and files are written in languages specific to the application, such as *gnuplot* (Williams 95), Tool control language (*Tcl*) (Osterhout 94) (Welch 94), hypertext markup language (.html) (Berners-Lee 94a, 94b) (Hughes 94) or operating system shell scripts.

All of the programs described here are available electronically without charge to Internet users via anonymous ftp. The accompanying public electronic distribution of this reference includes both source code and compiled executable programs. All programs have been tested under SGI operating system Irix 5.2, and robot code has also been tested under operating system OS-9 version 2.3. Goals of the underwater virtual world project include making source code, applications and interfaces easily available in order to encourage scientific collaboration and educational use. Future

work includes porting virtual world viewers to a variety of operating systems and computer architectures.

An NPS AUV hypermedia home page has been prepared to facilitate examination and download of the various underwater virtual world components. This home page is available for interaction using any number of Internet-based browsers, most notably *Mosaic* (NCSA 94a). Resources on the home page include the electronic distribution, installation and execution instructions, pointers to related work, images, plots, papers and a variety of other media. World-Wide Web (WWW) Uniform Resource Locator (URL) for the NPS AUV home page is:

ftp://taurus.cs.nps.navy.mil/pub/auv/auv.html

To learn more about the World-Wide Web, consult "Entering the World-Wide Web (WWW): A Guide to Cyberspace" (Hughes 94). This paper describes everything needed to start using *Mosaic*, accessing all types of information sources, and using hypermedia and the Internet. *Mosaic* is a hypermedia browser that is freely available for Unix machines running X windows, Macintoshes and personal computers (PCs) running Windows. There are also free line mode browsers (such as *lynx* and *www*) that let you browse hypertext documents in a simple text mode using *any* computer which has an internet connection (Montulli 94) (Berners-Lee 94b).

The source code in this software guide is fully documented and reasonably self-explanatory. Graphics and hydrodynamics programs are written in the ANSI standard ***C++*** language, robot execution code and networking code is written in the Kernighan and Richie variant of the ***C*** language, plotting programs use *gnuplot* version 3.5 scripting language, and hypertext pages are written in standard Hypertext Markup Language (*.html*) (NCSA 94b). Despite extensive documentation, however, one programmer's clearly-written code may well be undecipherable to another programmer. Users are strongly encouraged to review the companion dissertation which derives the formulae and explains the concepts behind the software (Brutzman 94), retrieve the latest version of the underwater virtual world distribution, or contact the author for questions that remain unanswered. No guarantees of any

kind are implied. Many problems are open and progress occurs on a frequent basis when conducting research on robots and virtual worlds.

Hopefully this work will stimulate and encourage other research efforts to build large-scale virtual worlds in compatibly-extendable ways. Interested readers are encouraged to examine the AUV home page for project updates, read the documentation, and to contact the author regarding new uses of the software and potential research collaboration.

II. INSTALLATION, EXECUTION, CREATION AND REMOVAL

A. Introduction

The following Installation and Execution Guide is intended to enable any new user to download, install, execute and understand the NPS AUV Underwater Virtual World. Users may want to solely download the viewer software to observe an Internet-wide mission exercise, or run the entire set of programs making up the virtual world. Users of the robot execution program can enter their electronic mail address and receive a mission report following each mission. A sample mission report shows the mission script used for the canonical SIGGRAPH 94 mission, a project abstract, pointers to related work and the execution orders generated by the mission script. Mission reports are also available by "fingering" the home auv account, e.g.

finger auv@dude.cs.nps.navy.mil

Finally, shell script files for archive creation and removal are included to document automated archive maintenance.

B. *auv-uvw.INSTALL* Installation and Execution Guide

```
//----------------------------------------------------------------------------//

    Installation & Execution Guide for the NPS AUV Underwater Virtual World

    auv-uvw.INSTALL                                                  19 OCT 94

    Don Brutzman                                         brutzman@nps.navy.mil

    URL: ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.virtual-world.INSTALL

//----------------------------------------------------------------------------//
```

Table of Contents


```
  This is a big project!  I've tried to make this guide as concise and readable
  as possible.  However some work is necessary to read and absorb this material.

  Suggestions on how to further streamline the guide are very welcome.  Thanks!

//----------------------------------------------------------------------------//

Distribution information

   For the latest greatest distribution, plus pointers to all of the supporting
   public domain programs, use Mosaic or a World-Wide Web line browser to
   connect to the NPS AUV home page:

        ftp://taurus.cs.nps.navy.mil/pub/auv/auv.html

   To retrieve the distribution directly:

        ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.tar.Z

   To receive a copy of a recent mission report:

     finger auv@dude.cs.nps.navy.mil

//----------------------------------------------------------------------------//

System requirements

   The auv-uvw 'viewer' requires SGI Irix 5.2 or better with OpenInventor 2.0
     Execution Only Environment (inventor_eoe) installed.  This is provided
     free with all versions of Irix, it merely needs to be installed.
     You can tell if it is installed by typing 'versions' at the SGI prompt.

   Modification and recompilation of the 'viewer' requires installation of
```

```
inventor_dev Inventor 3D Toolkit, 2.0 development option.
  The virtual world 'viewer' is written in C++ using OpenInventor
  graphics, and it also uses the NPSNET DIS 2.0.3 multicast libraries
  (which are also included as source/binaries in this distribution).

The 'dynamics' virtual world software has been tested under Irix 5.2,
  is written in C++, and also uses the NPSNET DIS 2.0.3 multicast
  libraries.  You should run it on a sound-equipped workstation for
  best results.

The robot 'execution' code has been tested under Irix 5.2 and OS-9 v2.3
  It is written in ANSI C with special #defines for OS-9 K&R C
  compatibility, allowing it to be run without modifications either under
  Irix or on the NPS Autonomous Underwater Vehicle 68030 microprocessor.
  It ought to be portable to most any plain vanilla C compiler.

//----------------------------------------------------------------------------//

Retrieving the distribution auv-uvw.tar.Z

  The easiest way is to retrieve & save the tar file is to point Mosaic at

    ftp://taurus.cs.nps.navy.mil/pub/mosaic/auv.html

  If instead you use anonymous ftp, here is a sample session:

    unix>> ftp taurus.cs.nps.navy.mil
    Connected to taurus.cs.nps.navy.mil.
    220 taurus FTP server (Version 6.10 Wed Mar 18 11:57:03 PST 1992) ready.

    Name (taurus.cs.nps.navy.mil:yourname): anonymous
    331 Guest login ok, send e-mail address as password.
    Password: your.email.address.here

    ftp> cd pub/auv
    250 CWD command successful.
    ftp> binary
    200 Type set to I.
    ftp> get auv-uvw.tar.Z
    local: auv-uvw.tar.Z remote: auv-uvw.tar.Z
    150 Opening BINARY mode data connection for auv-uvw.tar.Z(4747669 bytes)
    226 Transfer complete.
    4747669 bytes received in 34.32 seconds (135.08 Kbytes/s)
    ftp> quit
    221 Goodbye.

  The uncompressed distribution is about 10 MB, not counting the extra
    recommended applications.  The compressed distribution is about 5 MB.
      Precise distribution sizes can vary when changes are incorporated.

//----------------------------------------------------------------------------//

Installing the distribution auv-uvw.tar.Z

  Uncompress auv-uvw.tar.Z in your root directory by typing

    unix:yourhome>    uncompress auv-uvw.tar.Z

    unix:yourhome>    tar -xvf   auv-uvw.tar

  Backup & update your MBone session directory configuration file .sd.tcl

    unix:yourhome>    cp  .sd.tcl          .sd.tcl.old
```

```
unix:yourhome>    cp  .sd.tcl.auv-uvw  .sd.tcl
```

Edit the MBone tool aliases to include directory paths as appropriate.
They are located at the very end of the new file .sd.tcl

```
unix:yourhome>    zip .sd.tcl
```

After editing, exit and restart session directory (sd):

```
unix:yourhome>    sd
```

You can now run the viewer from an sd advertisment for the NPS AUV Underwater
Virtual World (if there is one running - this is optional).

You will notice some auv-uvw information files in your root directory
and several new directories:

```
yourhome/auv-uvw/ OpenInventor viewer program & data files

yourhome/execution/AUV (robot) execution level code and
                   mission scripts

yourhome/dynamics/Virtual world for robot: dynamics, sonar
                   and other components

yourhome/dynamics/speechText-to-speech audio files from mission
                   scripts.  These originate from queries
                   to the Say.. server
```

The auv-uvw distribution should now be fully installed.

//---//

MBone connection (optional but recommended)

Multicast Backbone (MBone) connectivity is needed if you want
to participate in remote exercises. A good way to test your network's
connectivity is to type 'sd' (session directory). If you get a menu
of MBone programs starting to slowly build up, you are in business.

To receive Distributed Interactive Simulation (DIS) robot execution
Protocol Data Units (PDUs) originating on your own network, you
do not have to be connected to the Internet-wide MBone.

SGI's Irix 5.2 operating system supports multicast in the kernel, so
you can still run the robot/virtual world programs on one machine &
the auv-uvw viewer on another. Only one process per machine can grab
the multicast port, thus at least 2 machines are needed to run
everything.

To connect your system to the MBone takes some work but is worth it.
You will need the involvement & support of your network administrator.

The following article tells why MBone is so great and how to connect:

```
ftp://taurus.cs.nps.navy.mil/pub/mbmg/mbone.ps    (PostScript) or
ftp://taurus.cs.nps.navy.mil/pub/mbmg/mbone.html  (hypertext)  or
ftp://taurus.cs.nps.navy.mil/pub/mbmg/mbone.txt   (ASCII text)
```

Other installation pointers are on the NPS MBone home page at

```
     ftp://taurus.cs.nps.navy.mil/pub/mosaic/mbone.html

    Over a thousand subnets worldwide are already connected to the MBone.
    There are lots of people willing to help.  In practice we've seen it
    take people between a few days and a few weeks to start using MBone,
    but once connected further work is rarely required.  Good luck!

//--------------------------------------------------------------------------//

Information files:

    auv-uvw.virtual-world.README    (a mission report)
    auv-uvw.virtual-world.INSTALL   (this file)

  others are listed in the mission report and the home page site

//--------------------------------------------------------------------------//

Viewing remotely-executing robot missions via the MBone

  You must have MBone connectivity installed on your network for remote
    experiment observation.  If the NPS AUV Underwater Virtual World
    is advertised in sd, and you have installed the distribution,
    just select the session and multicast ports are passed automatically
    to the viewer.

  Alternatively, you can start the viewer manually from the command line:

    unix:yourhome>    auv-uvw.viewer
   or    unix:yourhome/auv-uvw>viewer

  The following command line switches are available with viewer:

    -address 224.2.###.##multicast address as advertised in sd session
    -port    #####    multicast wb port as advertised in sd session

   The virtual world viewer should now run.

//--------------------------------------------------------------------------//

Running all of the virtual world components locally

  Running a robot mission takes three processes and at least two machines.
  viewer and dynamics must be on different hosts, execution can run anywhere.

  Notes regarding sd session advertisements are optional.

 * First is the 'viewer' to get a window into the virtual world.  Run this
   on your most capable Iris workstation - Reality Engines are nice!
   You can use or disable texturing based on the capabilities of the machine.

    unix1>  cd auv-uvw
    unix1>  viewer

    The following optional command line switches are available with viewer:

    -address 224.2.###.##multicast address as advertised in sd session
    -port #####        multicast wb port as advertised in sd session
       -texture-on
       -texture-off
       -printdialog    pop up print dialog box for screen snapshots
       -noprintdialog

    The virtual world viewer now runs.
```

* The 'dynamics' program is the virtual world connection for robot execution: Run it on a sound-equipped workstation for best results, otherwise ignore any sound-card-missing error messages. The 'dynamics' program is purely text-based so graphics capabilities are irrelevant.

```
unix2>  cd dynamics
unix2>   dynamics

        Dynamics classes test selections
                L loop_test_with_execution_level ();
                M Multicast parameter input
                ttl=15, group address=224.2.121.93, port=3111
                O Ocean current vector reset  <0, 0, 0>
                H Hmatrix/quaternion exerciser
                R Rotation of quaternion & Hmatrix using p q r
                D Defaults
                I Invert matrix test
                E dEad_reckon_test_with_execution_level
                P PDU_skip_interval change (from 1)
                T Toggle TRACE = 0
                C DIS_net_close ();
                Q Quit
        Enter choice:
```

Select L for loop test with execution level

The following command line switches are available with dynamics program:

```
-ttl      #        time to live (be careful!) recommend 16 or less
-address 224.2.###.##multicast address as advertised in sd session
-port     #####    multicast wb port as advertised in sd session
-loop              start looping automatically
```

* Finally, 'execution' is the robot execution level software. 'execution' runs through the file 'mission.script' and gets real world responses from the virtual world program 'dynamics'. 'execution' can run under Unix or real-time operating system OS-9.

```
unix3>   cd execution
unix3>   cp mission.script.the_one_you_want mission.script
unix3>   execution  remotehost unix1.host.name
```

Enter your e-mail in the execution process window when asked.

See the file mission.script.HELP for info on how to write mission scripts.

//--//

Plotting mission telemetry using gnuplot

```
unix3:yourhome>   cd execution
unix3:yourhome/execution/gnuplot  auv_plot.gnu
```

For fewer points plotted (1 per second) out of the same dataset:

```
unix3:yourhome/execution/gnuplot  auv_plot_1_second.gnu
```

or use a postscript viewer (xpsview or ghostview) to look at plots directly:

```
   unix3> xpsview   -wp -skipc -or landscape ~/execution/AUV_telemetry.ps &
or unix3> ghostview              -landscape ~/execution/AUV_telemetry.ps &
```

//--//

Recompiling virtual world component executables

auv-uvw directory

(Requires inventor_dev Inventor 2.0 development option installed.
See system requirements section for details).

```
unix3:yourhome/auv-uvw/>    make viewer
```

dynamics directory

```
unix3:yourhome/dynamics/>   make dynamics
```

execution directory

```
     unix3:yourhome/execution/>  copy Makefile.SGI Makefile
     unix3:yourhome/execution/>  make execution

or   OS-9: yourhome/execution/>  copy Makefile.OS9 Makefile
     OS-9: yourhome/execution/>  make execution
```

Details on recompilation and modification are included in all source files.

//--//

Killing multicast processes

Sometimes multicast processes (such as 'viewer' and 'dynamics') do not die cleanly because they have forked a second multicast process.
viewer is especially prone to this behavior when invoked from session directory (sd), since sd must put the viewer process into the background.

You may need to take special action to ensure the processes are killed.
The following instructions will work on SGI machines:

To put a process to sleep that refuses to let go of the command prompt:

type a ^Z (control-Z) in the runaway process window

To see what processes are running:

```
unix> ps
```

To see ALL processes that are running (including previous zombie processes):

```
unix> ps -ea
```

To kill a process:

```
unix> kill -9 #### where #### is the Process ID (PID)
```

An example log:

```
[runaway process, you hit the control-Z key to suspend it]
^Z
Suspended

unix>> ps
    PID TTY        TIME COMD
  16558 ttyq3      9:37 viewer
  11289 ttyq3      0:01 tcsh
  16581 ttyq3      0:00 viewer
  16902 ttyq3      0:00 ps
```

```
unix>> kill -9 16558 16581

unix>> ps
    PID TTY       TIME COMD
  11289 ttyq3     0:01 tcsh
  16904 ttyq3     0:00 ps
```

//--//

Removing the entire distribution:

Be sure to move any files that you want to save to different directories.

You do not have to remove the old distribution prior to updating to a new version. However you might want to, so that superceded files in old distributions are not left over.

To remove everything, run the shell script auv-uvw.virtual-world.REMOVE:

```
unix> source   auv-uvw.virtual-world.REMOVE
```

//--//

Future work (lots!)

Porting the OpenInventor viewer code to other architectures and porting other virtual world components to other architectures depends on availability of OpenInventor and multicast TCP/IP support. Porting is planned for 1995.

Other people are always welcome to use and extend this code. Please keep me informed of your efforts if you find it of value.

Other research collaborations to extend the underwater virtual world to build up a truly large-scale Internet-wide fully distributed virtual world are of particular interest.

Integrate Dr. Larry Ziomek's Recursive Ray Acoustics (RRA) sonar model.

Incorporate pre-processed terrain datasets for Monterey Bay.

Virtual Reality Modeling Language (VRML) - see the working group http://www.wired.com/vrml/

Dissertation "A Virtual World for an Autonomous Underwater Vehicle" is being written and will be made publicly available.

//--//

Contact information:


```
Don Brutzman                                                  brutzman@nps.navy.mil
Code OR/Br   Naval Postgraduate School   [Glasgow 204]   work (408) 656-2149
Monterey California 93943-5000  USA                          fax  (408) 656-2595
```


//--//

C. *mission.output.email* Mission Output Electronic Mail Report

The following report is sent by electronic mail to users who execute a robot mission. It demonstrates that robots, once networked, can report back results only when they are of interest or human intervention is specifically desired. The goal is for the robot to meet user directives conveniently, rather than requiring constant monitoring or supervision.

The mission report combines an NPS AUV reference information file, the mission script that drove the robot, and the resulting time line of propulsor and plane surface orders. It is also available by querying the robot account with the Unix "finger" command: *finger auv@cs.dude.nps.navy.mil*

```
                     NPS AUV Mission Report

Here is a mission report generated by the Naval Postgraduate School (NPS)
Autonomous Underwater Vehicle (AUV) running in an Underwater Virtual World.

This message was created by the operating robot connected to the network.
Actual robots with network connections can send reports such as this
to research teams and interested individuals when scientific discoveries
are made or unusual circumstances occur.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

virtually yours, the NPS AUV.          .          . . .. 8-/‾nps auv‾)
                          .            .          . . .. 8-\_________)

===========================================================================

The following Universal Resource Locators (URL's) are pointers that lead to
more information about the NPS AUV Underwater Virtual World & related work.

* The AUV underwater virtual world source code, information files and
* executable programs (for SGI Irix 5.2 machines) can be used to monitor
* NPS AUV DIS PDU's from anywhere on the Internet MBone.  It is freely
* available via anonymous ftp as a compressed tar file at
ftp://taurus.cs.nps.navy.mil/pub/auv/auv_uvw.tar.Z

* NPS AUV home page free software distribution, research summary and
* anonymous ftp directory:
ftp://taurus.cs.nps.navy.mil/pub/mosaic/auv.html
ftp://taurus.cs.nps.navy.mil/pub/auv/AUVPAPERS
ftp://taurus.cs.nps.navy.mil/pub/auv/

* Naval Postgraduate School World-Wide-Web (WWW) home page includes many
* additional pointers to the NPSNET Virtual Battlefield, DIS, MBone, and
* I3LA regional research around Monterey Bay:
ftp://taurus.cs.nps.navy.mil/pub/mosaic/nps_mosaic.html

* To learn about Internet audio & video on the Multicast Backbone (MBONE):
```

ftp://taurus.cs.nps.navy.mil/pub/mosaic/mbone.html

* How the robot voice was synthesized live from text over the Internet using
* Axel Belinfante's "Say..." speech server at University of Twente,
* Netherlands which uses Nick Ing-Simmons' phoneme synthesizer 'rsynth':
http://utis179.cs.utwente.nl:8001/say/?

* Postscript plots (20 pages) of vehicle telemetry during SIGGRAPH mission:
ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/AUV_telemetry.ps.Z

* The people from NPS involved in SIGGRAPH 94 _The Edge_ NPS AUV exhibit:
ftp://taurus.cs.nps.navy.mil/pub/pratts/home.html

* Learning more about the World-Wide-Web, URLs and Mosaic:
* "Entering the World-Wide Web: A Guide to Cyberspace" by Kevin Hughes

```
http://www.eit.com/web/www.guide/                       hypertext
ftp://ftp.eit.com/pub/web.guide/guide.61/guide.61.ps.Z  Postscript
ftp://ftp.eit.com/pub/web.guide/guide.61/guide.61.txt   text
```

* A recent copy of this mission report can be found by an auv account finger:
finger auv@dude.cs.nps.navy.mil

If you can't find taurus.cs.nps.navy.mil, it has IP number 131.120.1.13
Additional references are available on request.

==

This project is part of a PhD dissertation. Here is the abstract:

A Virtual World for an Autonomous Underwater Vehicle

Don Brutzman
Code OR/Br, Naval Postgraduate School
Monterey California 93943-5000 USA
(408) 656-2149 office, (408) 656-2595 fax
brutzman@nps.navy.mil

A critical bottleneck exists in Autonomous Underwater Vehicle (AUV) design and development. It is tremendously difficult to observe, communicate with and test underwater robots, because they operate in a remote and hazardous environment where physical dynamics and sensing modalities are counterintuitive.

An underwater virtual world can comprehensively model all salient functional characteristics of the real world in real time. This virtual world is designed from the perspective of the robot, enabling realistic AUV evaluation and testing in the laboratory. 3D real-time graphics are our window into that virtual world. Visualization of robot interactions within a virtual world permits sophisticated analyses of robot performance that are otherwise unavailable. Sonar visualization permits researchers to accurately "look over the robot's shoulder" or even "see through the robot's eyes" to intuitively understand sensor-environment interactions.

Distribution of underwater virtual world components enables scalability and real-time response. The IEEE Distributed Interactive Simulation (DIS) protocol is used for compatible live interaction with other virtual worlds. Network access allows individuals remote access. This is demonstrated via MBONE collaboration with others outside The Edge, and Mosaic access to pertinent archived images, papers, datasets, software, sound clips, text and any other computer-storable media.

This project presents the frontier of 3D real-time graphics for underwater robotics, ocean exploration, sonar visualization and worldwide scientific collaboration.

==

Questions, comments and collaborations are welcome. Please contact:

Don Brutzman brutzman@nps.navy.mil
Code OR/Br Naval Postgraduate School [Glasgow 204] work (408) 656-2149
Monterey California 93943-5000 USA fax (408) 656-2595

SIGGRAPH 94 Collaborators: Michael J. Zyda, Paul T. Barham, John S. Falby,
Anthony J. Healey, Shirley Isakari, Rodney Luck,
Michael R. Macedonia, Robert B. Mcghee,
David R. Pratt, Lawrence J. Ziomek
Naval Postgraduate School, Monterey California

==

Your AUV robot mission completed successfully. Here is the mission you ran:

```
# your mission is

# mission.script.rotate_test

time              0

timestep 0.1

# initialize vehicle

depth 45

position     0    0   45

orientation 0    0    0

thrusters-on

rotate  16

pause

wait 60

rotate 0

thrusters-off

wait 120

step

keyboard

keyboard-off

# mission complete

kill
```

```
# NPS AUV file mission.output.orders: commanded propulsion orders versus time
#
#         timestep:  0.10 seconds
#
# time  heading North East Depth    rpm   rpm    stern  stern   vertical      lateral
#                   x     y    z    port  stbd   plane  rudder  thrusters    thrusters
#                                                               bow/stern    bow/stern

   0.0      0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0   0.0   0.0   0.0   0.0
# timestep:  0.10 seconds
   0.0      0.0   0.0   0.0  45.0    0.0    0.0    0.0    0.0   0.0   0.0   0.0   0.0
  60.0      0.0   0.0   0.0  45.0    0.0    0.0    0.0    0.0   0.0   0.0   0.0   0.0
 180.1      0.0   0.0   0.0  45.0    0.0    0.0    0.0    0.0   0.0   0.0   0.0   0.0
```

D. *make_auv_uvw_tar* Archive Creation Script

```
# make_auv_uvw_tar:  NPS AUV Underwater Virtual World archive creation script

# Don Brutzman   20 OCT 94

echo create auv underwater virtual world distribution tar file auv-uvw.tar.Z

# Notes:
# do not make this tar on gravy1, it doesn't have enough memory for 'speech'
# ensure mission.script.siggraph was the latest (best) mission run
# this script is not included in the distribution.

# NPS Autonomous Underwater Vehicle (AUV) World-Wide Web Home Page
# ftp://taurus.cs.nps.navy.mil/pub/mosaic/auv.html

cd  /n/dude/work/brutzman

echo    'removing old files...'
rm  -f   auv-uvw.tar
rm  -f   auv-uvw.tar.Z

# Standard e-mail report becomes the README file
rm  -f                                    auv-uvw.virtual-world.README
cp      execution/mission.output.email    auv-uvw.virtual-world.README
chmod -w                                  auv-uvw.virtual-world.README
tar -cvf auv-uvw.tar                      auv-uvw.virtual-world.README
rcp     auv-uvw.virtual-world.README      taurus:~ftp/pub/auv
chmod +w                                  auv-uvw.virtual-world.README
rm  -f                                    auv-uvw.virtual-world.README

rm  -f                                    auv-uvw.virtual-world.INSTALL
cp      execution/auv-uvw.INSTALL         auv-uvw.virtual-world.INSTALL
chmod -w                                  auv-uvw.virtual-world.INSTALL
rcp     auv-uvw.virtual-world.INSTALL     taurus:~ftp/pub/auv
tar -rvf auv-uvw.tar                      auv-uvw.virtual-world.INSTALL
chmod +w                                  auv-uvw.virtual-world.INSTALL
rm  -f                                    auv-uvw.virtual-world.INSTALL

rm  -f                                    auv-uvw.virtual-world.REMOVE
cp      execution/auv-uvw.REMOVE          auv-uvw.virtual-world.REMOVE
chmod +x                                  auv-uvw.virtual-world.REMOVE
tar -rvf auv-uvw.tar                      auv-uvw.virtual-world.REMOVE
rm                                        auv-uvw.virtual-world.REMOVE

rcp     execution/mission.script.HELPtaurus:~ftp/pub/auv

tar -rvf auv-uvw.tar auv-uvw.viewer

tar -rvf auv-uvw.tar auv-uvw/viewer
tar -rvf auv-uvw.tar auv-uvw/viewer.C
tar -rvf auv-uvw.tar auv-uvw/Makefile
tar -rvf auv-uvw.tar auv-uvw/overhang.rgb
tar -rvf auv-uvw.tar auv-uvw/auv.iv
tar -rvf auv-uvw.tar auv-uvw/Jason.iv
tar -rvf auv-uvw.tar auv-uvw/Platform.iv
tar -rvf auv-uvw.tar auv-uvw/Testtank.iv
tar -rvf auv-uvw.tar auv-uvw/testtank.C

tar -rvf auv-uvw.tar auv.html
tar -rvf auv-uvw.tar mbone.html
tar -rvf auv-uvw.tar nps_mosaic.html

tar -rvf auv-uvw.tar .sd.tcl.auv-uvw
tar -rvf auv-uvw.tar .mailcap.auv-uvw
```

```
tar -rvf auv-uvw.tar .cshrc-excerpt.auv-uvw

tar -rvf auv-uvw.tar execution/execution
tar -rvf auv-uvw.tar execution/execution.c
tar -rvf auv-uvw.tar execution/parse_functions.c
tar -rvf auv-uvw.tar execution/globals.c
tar -rvf auv-uvw.tar execution/globals.h
tar -rvf auv-uvw.tar execution/defines.h
tar -rvf auv-uvw.tar execution/errno.h
tar -rvf auv-uvw.tar execution/modes.h
tar -rvf auv-uvw.tar execution/setsys.h
tar -rvf auv-uvw.tar execution/sgstat.h
tar -rvf auv-uvw.tar execution/Makefile*

chmod -w             execution/mission.script.HELP
tar -rvf auv-uvw.tar execution/mission.*
chmod 644            execution/mission.script.HELP

tar -rvf auv-uvw.tar execution/auv_plot.gnu
tar -rvf auv-uvw.tar execution/auv_plot_1_second.gnu
tar -rvf auv-uvw.tar execution/print_plots
tar -rvf auv-uvw.tar execution/disbridge.c
tar -rvf auv-uvw.tar execution/os9sender.c
tar -rvf auv-uvw.tar execution/os9server.c

tar -rvf auv-uvw.tar dynamics/dynamics
tar -rvf auv-uvw.tar dynamics/Makefile
tar -rvf auv-uvw.tar dynamics/*.H
tar -rvf auv-uvw.tar dynamics/*.C

echo 'remove speech/numbers.au originating from high replication counts...'
rm  -f               dynamics/speech/2?.au
rm  -f               dynamics/speech/3?.au
rm  -f               dynamics/speech/4?.au
rm  -f               dynamics/speech/5?.au
rm  -f               dynamics/speech/6?.au
rm  -f               dynamics/speech/7?.au
rm  -f               dynamics/speech/8?.au
rm  -f               dynamics/speech/9?.au
rm  -f               dynamics/speech/1??.au
rm  -f               dynamics/speech/2??.au
rm  -f               dynamics/speech/3??.au
rm  -f               dynamics/speech/4??.au
echo 'the speech directory can get very large and occasionally needs pruning...'
tar -rvf auv-uvw.tar dynamics/speech/*.au

tar -rvf auv-uvw.tar DIS.mcast/*

# The following files are provided separately to reduce tar file size.
#          See the auv home page for remote access instructions
#          ftp://taurus.cs.nps.navy.mil/pub/mosaic/auv.html
# tar -rvf auv-uvw.tar execution/AUV_telemetry.ps
# tar -rvf auv-uvw.tar aaai92ws.ps.Z
# tar -rvf auv-uvw.tar dynamics/www
# tar -rvf auv-uvw.tar execution/gnuplot

ls -l     auv-uvw.tar
echo 'compressing tar file...'
compress  auv-uvw.tar

ls -l     auv-uvw.tar.Z
echo 'rcp auv-uvw.tar.Z taurus:~ftp/pub/auv'
      rcp auv-uvw.tar.Z taurus:~ftp/pub/auv

echo "archive information on taurus anonymous ftp server:"
      rsh taurus 'ls -l ~ftp/pub/auv/auv-uvw.tar.Z'
echo 'make_auv_uvw_tar complete.'
```

E. *auv-uvw.virtual-world.REMOVE* Archive Removal Script

```
# auv-uvw.virtual-world.REMOVE  archive removal shell script

# To run this script, please type the following at the unix prompt:
#    chmod +x auv-uvw.virtual-world.REMOVE
#    source   auv-uvw.virtual-world.REMOVE

# Don Brutzman  19 OCT 94

# NPS Autonomous Underwater Vehicle (AUV) World-Wide Web Home Page
# ftp://taurus.cs.nps.navy.mil/pub/mosaic/auv.html

echo 'remove auv underwater virtual world distribution...'
echo 'be sure you run this from the directory you installed it'
echo '(ordinarily your home directory)'

rm  -f       auv-uvw.tar
rm  -f       auv-uvw.tar.Z

echo 'warning:  unaliasing rm...'
unalias rm

rm     -f -r execution/*
rm     -f -r dynamics/*
rm     -f -r auv-uvw/*
rm     -f -r DIS.mcast/*

# remove any inadvertantly created .dot files
rm     -f -r execution/.*
rm     -f -r dynamics/.*
rm     -f -r auv-uvw/.*
rm     -f -r DIS.mcast/.*

rmdir        execution
rmdir        dynamics
rmdir        auv-uvw
rmdir        DIS.mcast

rm  -f       auv-uvw.viewer
rm  -f       auv.html
rm  -f       mbone.html
rm  -f       nps_mosaic.html
rm  -f       .sd.tcl.auv-uvw
rm  -f       .mailcap.auv-uvw
rm  -f       .cshrc-excerpt.auv-uvw

rm  -f       auv-uvw.virtual-world.README
rm  -f       auv-uvw.virtual-world.INSTALL
rm  -f       auv-uvw.virtual-world.REMOVE
rm  -f       auv-uvw.*

echo 'restoring .sd.tcl with your saved .sd.tcl.old, please confirm...'

echo cp .sd.tcl.old .sd.tcl
     cp .sd.tcl.old .sd.tcl

echo 'auv-uvw.virtual-world.REMOVE complete.'
```

III. NPS AUTONOMOUS UNDERWATER VEHICLE EXECUTION LEVEL

A. Introduction

The execution level of robot software is the hard real-time process which controls propellers, thrusters, plane surfaces, sonars and other sensors. Hardware interfaces to the robot microprocessor are controlled by the execution level. The execution level is further tasked with interprocess communications with the tactical and strategic levels of the robot architecture (Byrnes 93), as well as the virtual world when operating in the laboratory.

The execution code presented here has been tested under a wide variety of laboratory conditions. Regrettably much of the interface code to actual vehicle hardware has not been tested since early 1994 when a battery explosion caused major damage to the NPS AUV II. Hardware repairs from February through October 1994 were required before in-water testing resumed. Nevertheless pre-explosion hardware interface source code has been retained in place and clearly identified through comments and compile flags. Future work includes verifying hardware interface code (Marco 95), updating the execution code and then running the combined virtual world/real world version of the execution level in the water.

An important benefit of a robot connected to an integrated simulator (Brutzman 92a, 92b, 92c) or underwater virtual world (Brutzman 94) is that robot software development can continue independently of actual robot readiness. Approximately half of the execution level source code presented here was developed during the eight month period that the NPS AUV II was out of commission. This advantage alone proved to be strong justification for the value of providing a network connection to the actual robot microprocessor and software. The essential requirement for integrated simulation in an underwater robotics research and development program has been subsequently confirmed by other researchers (Trimble 94) (Kuroda 94).

Another important strength of the robot software is that the Kernigan and Richie variant *C* language source code is able to compile correctly under two dissimilar operating systems: SGI Irix 5.2, a Unix variant, and OS-9 version 2.3, a real-time operating system that includes features similar to both Unix and DOS. Compiler directives, customized makefiles and command line aliases are used to provide this cross-platform portability. Network communication via sockets has also been made compatible between these two operating systems. This flexibility has been invaluable for rapid and effective software development. One example out of many is that diagnosis and debugging tools under Unix are far more robust than under OS-9, permitting detection and correction of subtle difficulties not detectable by the OS-9 *C* language compiler and linker.

Future work includes upgrading execution and tactical level communication links. Currently communications between levels use serial and parallel ports, a highly customized and error-prone hardware arrangement. By adding Ethernet connectivity to the tactical level computer, all software levels will be able to communicate via sockets regardless of what networked processor or workstation they might physically reside on. Additional long term goals include extending Internet protocol compatibility to acoustic telemetry communications. Ultimately underwater robots will be independent nodes on the Internet regardless of whether they are in the laboratory, operating with a tether or swimming freely and autonomously within an acoustic local area network (ALAN). The sophistication of virtual worlds will grow closer to that of the real world as robots and researchers operate in each.

B. *mission.script.HELP* Robot Execution Level Mission Script Syntax

```
//----------------------------------------------------------------------------//

        ~/execution/mission.script.HELP                     11 October 94

        Mission script syntax for NPS AUV execution level control in

                              the NPS AUV  Underwater  Virtual World

        Don Brutzman                                  brutzman@nps.navy.mil

//----------------------------------------------------------------------------//

This file describes how to change and create NPS AUV mission script files.
Files and the 'execution' program are in the ~/execution subdirectory.

        To run a new mission, copy an existing mission file over file
         'mission.script' or edit the mission.script file for a new mission.

        Example:        unix> cd execution
                        unix> cp  mission.script.siggraph  mission.script
                        unix> execution  remotehost fletch.cs.nps.navy.mil

Script commands are read by AUV execution level (execution.c)
        from the "mission.script" default file at the start of each mission.

Some of the following commands will also work when invoked from the command
        line upon execution.

Here are script keywords (and synonyms) that are currently recognized:

         |            |
---------+------------+-------------------------------------------------------
Keywords | Parameters |  Description
Synonyms | [optional] |  (units in feet or degrees as appropriate)
---------+------------+-------------------------------------------------------
         |            |
HELP                    Provide a list of available keywords
?                       (as specified in this HELP file).
/?
-?

REMOTEHOST  hostname    tells execution level to open socket to virtual world
REMOTE      hostname    which is already executed and waiting on 'hostname'
                        REMOTEHOST is a command line switch.  Example:
                        unix> execution  remotehost fletch.cs.nps.navy.mil

//                      comments follow on this line which are not executed
/*                      note comments will still be spoken if AUDIO-ON
#                       pound sign also indicates a comment if in first column

WAIT                    #       Wait (or run) for # seconds (letting the robot
execute)
RUN             #       prior to reading from the script file again

TIME            #       Wait (or run) until robot clock time #
WAITUNTIL       #       (letting the robot execute its current orders)
PAUSEUNTIL      #       prior to reading from the script file again

TIMESTEP        #       change default execution level time step interval
TIME-STEP       #       from default of 0.1 sec to # sec

PAUSE                   temporarily stop execution until <enter> is pressed
```

REALTIME REAL-TIME	run execution level code in real-time (busy wait at the end of each timestep if time remains)
NOREALTIME NO-REALTIME NONREALTIME NOWAIT NO-WAIT NOPAUSE NO-PAUSE	run execution level code as quickly as possible
MISSION filename SCRIPT filename FILE filename	Replace 'mission.script' with 'filename' and start the new mission
KEYBOARD KEYBOARD-ON	read script commands from keyboard
KEYBOARD-OFF NO-KEYBOARD	read script commands from mission.script file
QUIT STOP DONE EXIT REPEAT RESTART COMPLETE <eof> marker	do not execute any more commands in this script, but repeat the mission again if LOOP-FOREVER is set
KILL SHUTDOWN	same as QUIT but also shuts down socket to virtual world 'dynamics' process.
RPM # [##] SPEED # [##] PROPS # [##] PROPELLORS # [##]	Set ordered rpm values to # for both propellers [or independently set left & right rpm values to # and ## respectively] maximum propellor speed is +- 700 rpm => 2 ft/sec
COURSE # HEADING YAW #	Set new ordered course (commanded yaw angle) #
TURN # CHANGE-COURSE #	Change ordered course by # degrees (positive # to starboard, negative # to port)
RUDDER #	Force rudder to remain at # degrees
DEADSTICKRUDDER [#]	Force rudder to remain at 0 [or #] degrees
DEPTH #	Set new ordered depth (commanded z)
PLANES #	Force planes to remain at # degrees
DEADSTICKPLANES [#]	Force planes to remain at 0 [or #] degrees
THRUSTERS-ON THRUSTERS THRUSTERON THRUSTERSON	Enable vertical and lateral thruster control
NOTHRUSTER NOTHRUSTERS THRUSTERS-OFF THRUSTERSOFF	Disable vertical and lateral thruster control
ROTATE #	open loop lateral thruster rotation control at # degrees/sec

NOROTATE ROTATEOFF ROTATE-OFF		disable open loop lateral thruster rotation control
LATERAL		# open loop lateral thruster translation control at # ft/sec (positive is to starboard, maximum is 0.5 ft/sec)
NOLATERAL LATERALOFF LATERAL-OFF		disable open loop lateral thruster translation control
POSITION LOCATION	# ## ### # ## ###	reset vehicle dead reckon position to (x, y, z) = (#, ##, ###)
ORIENTATION ROTATION	# ## ### # ## ###	reset vehicle orientation to (phi, theta, psi) = (#, ##, ###)
CONTINUE GO		continue reading script & executing, no action performed
STEP SINGLE-STEP		loop for another timestep prior to reading script again. Particularly useful in keyboard mode.
TRACE TRACE-ON		enable verbose print statements in execution level
TRACEOFF TRACE-OFF NOTRACE NO-TRACE		disable verbose print statements in execution level
LOOPFOREVER LOOP-FOREVER		repeat current mission when done. each repetition is called a 'replication.'
LOOPONCE LOOP-ONCE		do not LOOPFOREVER, stop when end of script is reached
LOOPFILEBACKUP LOOP-FILE-BACKUP		back up output files during each loop replication to permit inspection while new files are written the backup files are in execution directory: output.telemetry.previous & output.1_second.previous
ENTERCONTROLCONSTANTS ENTER-CONTROL-CONSTANTS		start a keyboard dialog to enter revised control algorithm coefficients
SLIDINGMODECOURSE SLIDING-MODE-COURSE		Sliding mode course control algorithm (not yet working)
SLIDINGMODEOFF SLIDING-MODE-OFF		Disable sliding mode course control algorithm
SONARTRACE		enable verbose print statements in execution sonar code
SONARTRACEOFF		disable verbose print statements in execution sonar code
SONARINSTALLED		sonar interface hardware cards are installed, use them
PARALLELPORTTRACE		enable trace statements for parallel port communications
AUDIBLE AUDIO AUDIO-ON SOUND-ON SOUNDON SOUND		enable text-to-speech audio output

SILENT SILENCE NOSOUND SOUNDOFF SOUND-OFF AUDIOOFF AUDIO-OFF QUIET	disable text-to-speech audio output
EMAIL EMAIL-ON E-MAIL E-MAIL-ON EMAILON	ask user for electronic mail address at mission start, send user an electronic mail report at mission finish
EMAILOFF EMAIL-OFF E-MAILOFF E-MAIL-OFF NO-E-MAIL NO-EMAIL NO-E-MAIL NOEMAIL	disable electronic mail address query feature
WAYPOINT #X #Y [#Z] WAYPOINT-ON #X #Y [#Z]	Point towards waypoint with coordinates (#X, #Y) (depth #Z optional). Leave waypoint control by ordering course, rudder, sliding-mode, rotate or lateral thruster control.
WAYPOINTFOLLOW WAYPOINT-FOLLOW	Set mode to arrive at each waypoint before reading the next mission script command, i.e. continue towards each waypoint for however long it takes to reach the standoff distance before pausing to read the next command.
WAYPOINTFOLLOWOFF WAYPOINT-FOLLOW-OFF	Disables WAYPOINTFOLLOW mode
STANDOFF # STAND-OFF # STANDOFFDISTANCE # STANDOFF-DISTANCE # STAND-OFF-DISTANCE #	Change standoff distance for WAYPOINT-FOLLOW and HOVER control
HOVER [#X #Y [#Z]]	Hover using thrusters and propellers for longitudinal and lateral positioning at specified or previous waypoint
HOVER #X #Y #Z	[#orientation] [#standoff-distance] Uses WAYPOINT control until within #standoff-distance of HOVER point (#X, #Y, #Z), then switches to HOVER control with [optional] final #orientation Full speed (700 RPM) port & starboard is used if AUV distance to WAYPOINT is > #standoff-distance + 10', then slows to 200 RPM until within #standoff-distance, then HOVER control.

C. *execution.c* Robot Execution Level Real-Time Control Program

```
/**********************************************************************************/
/*
 Program:           execution.c  AUV execution level program

 Authors:           Don Brutzman, Dave Marco & Walt Landaker

 Revised:           28 October 94

 System:            AUV Gespac 68020/68030, OS-9 version 2.4
 Compiler:          Gespac cc Kernighan & Richie (K&R) C

 Compilation:       ftp>      put execution.c
                    auvsim1> chd execution
     [68020]        auvsim1> make -k2f execution
     [68030]        auvsim1> make      execution

     [Irix ]        fletch>  make execution

 Execution:
     [Irix ]        fletch> cd execution
                    fletch>    execution   remote dynamics-hostname

                    where dynamics-hostname is the IP name of the host running
                    the dynamics (virtual world) program

 Plotting:          see gnuplot scripts 'auv_plot.gnu' and 'auv_plot_1_second.gnu'
                    fletch> gnuplot  auv_plot.gnu

 Debugging:         gravy1:~brutzman/execution>> lint -lm execution.c

                    lint -lm -Iglobals.h -Idefines.h globals.c parse_functions.c \
                         execution.c

                    fletch>  make warnings

 Description:       closed loop for operation during vehicle in-water
                              missions as well as in virtual world

 Active changes:    Don Brutzman      working lab/virtual world networked version
                                    & tactical level interface

                    Dave Marco        working vehicle code
                                    & interfacing physical devices

 Future work:       Update digital <==> analog access for new vehicle hardware

                    Retest code after vehicle repaired

                    Sonar/altimeter integration code reintegrated/retested

                    Audio strings seem to be generated differently by OS-9

                    standardize parsing of command line and script commands

                    finish sliding mode control

                    change serial/parallel comms to sockets once
                        tactical level gets an Ethernet card

Testing interprocessor connections:
```

```
  parallel port    /P          OS-9 auvsim1> mfi_a3

                   LPT1:       DOS  auvsim2> portfix
                                             > print filename.txt

    serial port    (/T1) /TT   OS-9 auvsim1> wr2t1              then write text
                               OS-9 auvsim1> rdt1a              then read  text
                               DOS  auvsim2> C:\COMM\PROCOMM
                                                   then <alto> for chat mode
                                                  <altF10> help, <altX> exit
 Interfaces:

   Telemetry sent          via serial   port /tt [== /t1 at high baud rate]
   Telemetry received      via parallel port /P

   Telemetry is optionally passed to/from tactical level running on 80386

   Reads files:            mission.init  [mission initialization data file   ]
   Writes files:           output.data   [vehicle telemetry state vector data]
                           output.auv    [tactical order/executive report log]

   Sonar commands/replies via device port /t3

   Note that %F double formats are used instead of %lf on scanf() and sscanf()
             calls for OS-9 compatibility

*

/*****************************************************************************/

#include "globals.h"
#include "defines.h"

/*****************************************************************************/
/* function prototypes                                                        */

/* is there some way to put parameter specifications in the prototypes??     */
/*    only if we buy the ANSI C compiler from Microware (or shift to VxWorks) */

  void           closed_loop_control_module          ();
  void           get_control_constants               ();
  double         depth                               ();
  double         calculate_psi                       ();
  void           zero_gyro_data                      ();
  void           initialize_dacs                     ();
  void           initialize_adcs                     ();
  void           control_surface                     ();
  void           rudder                              ();
  void           planes                              ();
  void           main_motors_off                     ();
  void           alive                               ();
  void           record_data_on                      ();
  void           record_data_off                     ();
  void           record_data                         ();
  double         roll_angle                          ();
  double         pitch_angle                         ();
  double         heading                             ();
  double         roll_rate_gyro                      ();
  double         pitch_rate_gyro                     ();
  double         yaw_rate_gyro                       ();
  double         stbd_motor_rpm                      ();
  double         port_motor_rpm                      ();
  double         get_speed                           ();
  void           get_init_avg                        ();
  void           get_avg_rng                         ();

  void            open_device_paths                  ();
```

```
    void            close_device_paths              ();
    void            read_parallel_port              ();
    void            dac1                            ();
    void            dac2b                           ();
    int             adc1                            ();
    int             adc2                            ();
    void            Init_PortA                      ();
    void            Init_PortB                      ();
    unsigned char   Read_PortA                      ();
    unsigned char   Read_PortB                      ();
    unsigned short  Read_PortAB                     ();
    void            set_bsyA                        ();
    void            rst_bsyA                        ();
    int             ck_sta                          ();

    void            center_sonar                    ();
    char            query_sonar_1_reply             ();
    void            set_step_size                   ();
    void            tty_mode                        ();

    void            print_valid_keywords            ();

    void            open_virtual_world_socket       ();
    void            shutdown_virtual_world_socket   ();
    void            send_data_on_virtual_world_socket ();
    void            get_stream_from_virtual_world_socket ();

    double          degrees                         ();
    double          radians                         ();
    double          normalize                       ();
    double          normalize2                      ();
    double          radian_normalize                ();
    double          radian_normalize2               ();
    void            clamp                           ();

    double          atan2                           ();
    double          sinh                            ();
    double          cosh                            ();
    double          tanh                            ();

    double          sign                            ();

/* from parse_functions.c                                                   */

extern void         parse_command_line_flags        ();
extern void         parse_mission_script_commands   ();
extern void         parse_mission_string_commands   ();


/***************************************************************************/
/* OS-9 - specific function compatibility                                  */

#if defined (sgi)

void    tsleep   (unsigned svalue) { /* null body */}

void    _sysdate (format, time, date, day, tick)
                  int format, *time, *date, *tick; short *day;
                 { /* null body */}

double  pow      (xx, yy)       /* power function */
                  double xx, yy;
                 {return exp (yy * log (xx));}

int     _gs_rdy  (path) int path; { return 0; } /* bytes waiting on path */

int     _gs_opt  (path, buffer)
```

```
                int path; struct sgbuf *buffer;
               { return 0; }

int      _ss_opt  (path, buffer)
                int path; struct sgbuf *buffer;
               { return 0; }
#endif

/****************************************************************************/


/****************************************************************************/
/****************************************************************************/

main (argc, argv)

  int argc; char **argv;     /* command line arguments */
{
   if (TRACE && DISPLAYSCREEN) printf ("[start main:  execution]\n");

   strcpy (virtual_world_remote_host_name, VIRTUAL_WORLD_REMOTE_HOST_NAME);

   dt = TIMESTEP;

   parse_command_line_flags (argc, argv);

   get_control_constants ();

   initialize_dacs ();
   initialize_adcs ();

   open_device_paths ();

   record_data_on ();                   /* Open files for data logging       */

   open_virtual_world_socket ();        /* open connection to virtual world  */

   if (strlen (buffer) > 0)
      send_data_on_virtual_world_socket ();    /* SILENT? send to sound driver */

   strcpy (buffer, " A U V virtual world socket is open");
   send_data_on_virtual_world_socket (); /* buffer containing message sent */

   /*************************************************************************/
   do     /* while (LOCATIONLAB && LOOPFOREVER)                              */
   {      /*           indefinite repeat loop for long-duration lab testing  */
          /*----------------------------------------------------------------*/

    if (DISPLAYSCREEN)
    {
      if (LOCATIONLAB && (EMAIL == TRUE))
      {
        strcpy (buffer, " Please Enter Your E-mail Address");
        send_data_on_virtual_world_socket (); /* buffer containing msg sent */
        printf ("%s *** HERE ***:  ", buffer);
        strcpy (email_address, "");
        gets (email_address);
        sprintf (buffer, "Thanks %s\n", email_address);
        printf ("%s\n", buffer);
        send_data_on_virtual_world_socket (); /* buffer containing msg sent */

        if ((int) (strlen (email_address) > 2) &&
                  (strcmp (email_address, "brutzman") != 0) &&
                  (strcmp (email_address, "BRUTZMAN") != 0))
        {
             emailaddressfile = fopen (EMAILADDRESSFILENAME, "a");  /* append */
             fprintf (emailaddressfile, "%s\n", email_address);
```

```
        fclose  (emailaddressfile);
    }
  }
  else if (LOCATIONLAB == FALSE)
  {
    alive (10, start_dwell); /* Wag fin every 10 seconds for total duration
                                of start_dwell seconds                       */
    printf("  Position AUV for Directional Gyro Offset Measurement\n");
    printf("  Rate Gyro zero measurement\n");
    printf("  Hit <Enter> on AUV When Ready  *** Here ! ***\n");
    answer = getchar (); /* pause */
  }
  printf("  OK!!  Starting the mission.\n");
}
parse_mission_script_commands ();      /* read initial script orders      */

zero_gyro_data ();                     /* Get daily zeros for gyros       */

if (SONARINSTALLED) center_sonar ();   /* must have open_device_paths 1st */

strcpy (buffer, " ,,,,"); /* pause */
send_data_on_virtual_world_socket ();   /* buffer containing msg sent */
strcpy (buffer, " A U V is starting");
send_data_on_virtual_world_socket ();   /* buffer containing msg sent */

 /* Initialization of closed loop parameters                                */

 buffer_index            = 0;
 telemetry_records_saved = 0;
 mission_leg_counter     = 0;
 end_test                = FALSE;
 wrap_count              = 0;
 t                       = 0.0;

 /*-----------------------------------------------------------------------*/
 /*   Main program operational loop code                                  */

 if (TRACE && DISPLAYSCREEN)
     printf ("[Starting main program operational loop code... ]");

 previousloopclock = clock ();

 /*-----------------------------------------------------------------------*/
 while (end_test == FALSE) /* this is the realtime main operational loop */
                           /* when end_test == TRUE then loop is done     */
 {

    closed_loop_control_module (); /* closed loop code is here <<<<<<<<<<*/

 }                              /* end of real-time main operational loop */
 /*-----------------------------------------------------------------------*/

 /* lab version may repeat forever for long-duration testing:             */
 replication_count ++;

 if (LOCATIONLAB && LOOPFOREVER && DISPLAYSCREEN)
 {
     printf ("\n[LOOP FOREVER enabled, next loop is replication %d...]\n",
              replication_count);
     sprintf (buffer, " LOOP FOREVER enabled, next loop is replication");
     send_data_on_virtual_world_socket ();     /* buffer msg sent */
     sprintf (buffer, " %d", replication_count);
     send_data_on_virtual_world_socket ();     /* buffer msg sent */
 }

 if (LOCATIONLAB && LOOPFOREVER)
 {
```

```
        /* reset amount of time to wait for next command */
        time_next_command = 0.0;
        t                 = 0.0;
        if (DISPLAYSCREEN)
        {
            printf ("\nLoopforever reset time:  [time_next_command = 0.0] ");
            printf ("  [t = 0.0]\n");
        }
    }

    if (LOOPFILEBACKUP)
    {
        record_data_off ();

#if defined(sgi)
        printf ("rm                           output.telemetry.previous\n");
        system ("rm                           output.telemetry.previous"  );
        printf ("cp   mission.output.telemetry output.telemetry.previous\n");
        system ("cp   mission.output.telemetry output.telemetry.previous"  );
        printf ("rm                           output.1_second.previous\n" );
        system ("rm                           output.1_second.previous"   );
        printf ("cp   mission.output.1_second  output.1_second.previous\n" );
        system ("cp   mission.output.1_second  output.1_second.previous"   );
#else
        printf ("del                          output.telemetry.previous\n");
        system ("del                          output.telemetry.previous"  );
        printf ("copy mission.output.telemetry output.telemetry.previous\n");
        system ("copy mission.output.telemetry output.telemetry.previous"  );
        printf ("del                          output.1_second.previous\n" );
        system ("del                          output.1_second.previous"   );
        printf ("copy mission.output.1_second  output.1_second.previous\n" );
        system ("copy mission.output.1_second  output.1_second.previous"   );
#endif
        if (LOCATIONLAB)
        {
            strcpy (buffer, " telemetry data backup complete");
            send_data_on_virtual_world_socket ();    /* buffer msg sent */
        }
    }
    else  /* don't bother backing up most recent results                    */
    {
        if (LOCATIONLAB && LOOPFOREVER)
        {
            rewind (auvtextfile);
            rewind (auvdatafile);
            if (TRACE && DISPLAYSCREEN)
                printf ("[auvtextfile & auvdatafile rewound to ");
                printf ("output.data.previous & output.auv.previous]\n");
            strcpy (buffer, " telemetry data backup skipped");
            send_data_on_virtual_world_socket ();    /* buffer msg sent */
        }
    }

    fflush (auvscriptfile);
    if    (fclose (auvscriptfile) == 0)
    {
          if (DISPLAYSCREEN)
           printf ("[success closing auvscriptfile mission.script.backup]\n");
    }
    else  if (DISPLAYSCREEN)
           printf ("[failure closing auvscriptfile mission.script.backup]\n");

    fflush (auvordersfile);
    fclose (auvordersfile);

/*
#if defined(sgi)
```

```
     if (TRUE && DISPLAYSCREEN)
         printf ("rm                                   mission.output.orders\n");
     system     ("rm                                   mission.output.orders"  );
         printf ("mv   mission.output.orders.backup mission.output.orders\n");
     system     ("mv   mission.output.orders.backup mission.output.orders"  );
#else
         printf ("del                                  mission.output.orders\n");
     system     ("del                                  mission.output.orders"  );
         printf ("copy mission.output.orders.backup mission.output.orders\n");
     system     ("copy mission.output.orders.backup mission.output.orders"  );
         printf ("del  mission.output.orders.backup\n\n");
     system     ("del  mission.output.orders.backup"  );
#endif
*/

     /* - - - - - - - - - - - - - - orders - - - - - - - - - - - - - - - - - - - - */

#if defined(sgi)
  sprintf (buffer, "rm %s\n",                                         AUVORDERSFILENAME);
#else
  sprintf (buffer, "del %s\n",                                        AUVORDERSFILENAME);
#endif
  if (DISPLAYSCREEN) printf  ("%s\n", buffer);
  system     (buffer);

#if defined(sgi)
  sprintf (buffer, "mv   %s.backup %s\n", AUVORDERSFILENAME, AUVORDERSFILENAME);
#else
  sprintf (buffer, "copy %s.backup %s\n", AUVORDERSFILENAME, AUVORDERSFILENAME);
#endif
  if (DISPLAYSCREEN) printf  ("%s\n", buffer);
  system     (buffer);

#if defined(sgi)
#else
  sprintf (buffer, "del  %s      \n",                                 AUVORDERSFILENAME);
  if (DISPLAYSCREEN) printf  ("%s\n", buffer);
  system     (buffer);
#endif


     /* - - - - - - - - - - - - - - e-mail - - - - - - - - - - - - - - - - - - - - */

#if defined(sgi)
     sprintf (buffer, "rm %s\n",                                      AUVEMAILFILENAME);
#else
     sprintf (buffer, "del %s\n",                                     AUVEMAILFILENAME);
#endif
     if (DISPLAYSCREEN) printf  ("%s\n", buffer);
     system     (buffer);

#if defined(sgi)
     sprintf (buffer, "cp   %s    %s\n", AUVINFOFILENAME,    AUVEMAILFILENAME);
#else
     sprintf (buffer, "copy %s    %s\n", AUVINFOFILENAME,    AUVEMAILFILENAME);
#endif
     if (DISPLAYSCREEN) printf  ("%s\n", buffer);
     system     (buffer);

#if defined(sgi)
     sprintf (buffer, "cat  %s >> %s\n", AUVSCRIPTFILENAME, AUVEMAILFILENAME);
#else
     sprintf (buffer, "list %s >> %s\n", AUVSCRIPTFILENAME, AUVEMAILFILENAME);
#endif
     if (DISPLAYSCREEN) printf  ("%s\n", buffer);
     system     (buffer);
```

```
#if defined(sgi)
      sprintf (buffer, "cat  %s >> %s\n", AUVORDERSFILENAME, AUVEMAILFILENAME);
#else
      sprintf (buffer, "list %s >> %s\n", AUVORDERSFILENAME, AUVEMAILFILENAME);
#endif
      if (DISPLAYSCREEN) printf  ("%s\n", buffer);
      system     (buffer);

      if ((int) (strlen (email_address) >= 3) && (EMAIL == TRUE))
      {
         sprintf (buffer, "mail %s < %s", email_address,     AUVEMAILFILENAME);
#if defined(sgi)
         printf ("%s\n", buffer);
         system     (buffer);
#else
      /* system     (buffer);          /* e-mail not available directly on OS-9 */
         send_data_on_virtual_world_socket ();     /* buffer msg sent anyway */
#endif
      }

      /* permit changing the vehicle mission during continuous lab testing     */
      if (LOCATIONLAB && LOOPFOREVER)
      {
         get_control_constants ();
         previousloopclock = clock ();
         record_data_on ();

         strcpy (buffer, " Load mission again");
         send_data_on_virtual_world_socket ();
             /* buffer containing message sent */
      }

   } while (LOCATIONLAB && LOOPFOREVER); /* end of lab infinite loop (if any) */
   /*************************************************************************/

   main_motors_off (); /* all done, turn them off */

   if (TRACE && DISPLAYSCREEN)
       printf ("[sending 'shutdown' message to virtual world dynamics]\n");

   strcpy (buffer, "shutdown");             /* must start with 'shutdown' to die */
   send_data_on_virtual_world_socket (); /* buffer containing message sent */

   shutdown_virtual_world_socket ();      /* close connection to virtual world */

   close_device_paths ();

   record_data_off ();

   if (TRACE && DISPLAYSCREEN)
       printf ("[finishing main:  fflush (stdout), fflush (stderr)]\n");

   fflush (stdout);
   fflush (stderr);

   if (TRACE && DISPLAYSCREEN) printf ("[main exit:  return (0)]\n");

#if defined (sgi)
   if (DISPLAYSCREEN) printf ("gnuplot auv_plot_1_second.gnu\n");
   system ("gnuplot auv_plot_1_second.gnu"); /* display plotted results */
#endif

   return (0); /* main program exit */

} /* end main program block, execution is complete */

/****************************************************************************/
```

```
/*****************************************************************************/

void closed_loop_control_module ()
{
     if (TRACE && DISPLAYSCREEN)
         printf ("[start  closed_loop_control_module]\n");

     /* Speed Control ****************************************************/

     speed = get_speed ();

     rpm = (port_rpm_command + stbd_rpm_command) / 2.0;

     clamp (& rpm, 700.0, -700.0, "rpm");   /* bound maximum RPM */

     if (TRACE && DISPLAYSCREEN)
         printf ("[clamp (& rpm, 700.0, -700.0, \"rpm\") complete]\n");

     /* Main_Motor RPM Control *******************************************/

     /* note thruster use does not preclude propeller use                  */

     if (LOCATIONLAB == TRUE)
     {
        port_rpm = port_rpm_command;
        stbd_rpm = stbd_rpm_command;
     }
     else /* in water, propeller control */
     {
        port_rpm = port_motor_rpm ();
        stbd_rpm = stbd_motor_rpm ();
     }

if (NOT_YET_REIMPLEMENTED)
{
     main_motor_delta1 = fabs(rpm) - stbd_rpm;
     main_motor_delta2 = fabs(rpm) - port_rpm;

     /* this is reset windup for proportional integral control of motor speed */
     /*    in order to prevent accumulating the integral of speed error       */
     if (main_motor_delta1>50.0)  main_motor_delta1 = 50.0;
     if (main_motor_delta2>50.0)  main_motor_delta2 = 50.0;

     main_motor_volt1 = main_motor_volt1 +(rpm/fabs(rpm)*0.2*main_motor_delta1);
     main_motor_volt2 = main_motor_volt2 +(rpm/fabs(rpm)*0.2*main_motor_delta2);

     if (main_motor_volt1 > 1023) main_motor_volt1 = 1023;
     if (main_motor_volt1 < 0)    main_motor_volt1 = 0;
     if (main_motor_volt2 > 1023) main_motor_volt2 = 1023;
     if (main_motor_volt2 < 0)    main_motor_volt2 = 0;

     dac1(main_motor_volt1,RIGHT_MOTOR);
     dac1(main_motor_volt2,LEFT_MOTOR);
}
     /* if using virtual world dynamics, network is source of values <<<<<<<< */

        phi   = roll_angle      ();     /* read roll angle  */
        theta = pitch_angle     ();     /* read pitch angle */
        psi   = calculate_psi   ();     /* Read heading     */

        p     = roll_rate_gyro  ();     /* read roll rate   */
        q     = pitch_rate_gyro ();     /* read pitch rate  */
        r     = yaw_rate_gyro   ();     /* Read yaw rate    */
        z     = depth           ();     /* Read depth       */

     /* note:  in laboratory using virtual world, values above are superceded */
```

```
/* Control laws ****  NOTE:  all k_ constants  must be (+) positive **** */

waypoint_distance = sqrt (   (x - x_command) * (x - x_command)
                           + (y - y_command) * (y - y_command));

/* use WAYPOINTCONTROL (not HOVERCONTROL) until within standoff_distance */
if   ((HOVERCONTROL == TRUE) && (waypoint_distance > standoff_distance))
{
     WAYPOINTCONTROL  = TRUE;
     DEADSTICKRUDDER  = FALSE;
     if   (waypoint_distance > standoff_distance + 10.0)
     {
         port_rpm_command = 700;
         stbd_rpm_command = 700;
         fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                         t, psi_command, x_command, y_command, z_command,
                       port_rpm_command, stbd_rpm_command,
                         rudder_command,   planes_command,
                               bow_lateral_thruster_command,
                             stern_lateral_thruster_command,
                              bow_vertical_thruster_command,
                            stern_vertical_thruster_command);
     }
     else
     {
         port_rpm_command = 200;
         stbd_rpm_command = 200;
         fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                         t, psi_command, x_command, y_command, z_command,
                       port_rpm_command, stbd_rpm_command,
                         rudder_command,   planes_command,
                               bow_lateral_thruster_command,
                             stern_lateral_thruster_command,
                              bow_vertical_thruster_command,
                            stern_vertical_thruster_command);
     }
}
else if ((HOVERCONTROL == TRUE) && (WAYPOINTCONTROL == TRUE))
     /* restore proper HOVERCONTROL */
{
     WAYPOINTCONTROL  = FALSE;
     DEADSTICKRUDDER  = TRUE;
     rudder_command   = 0.0;
     psi_command      = psi_command_hover;
     port_rpm_command = 0;
     stbd_rpm_command = 0;
         fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                         t, psi_command, x_command, y_command, z_command,
                       port_rpm_command, stbd_rpm_command,
                         rudder_command,   planes_command,
                               bow_lateral_thruster_command,
                             stern_lateral_thruster_command,
                              bow_vertical_thruster_command,
                            stern_vertical_thruster_command);
}

if (WAYPOINTCONTROL == TRUE)
{
   /* note that a reversed x,y calling sequence is necessary */
   /*   in order to get correct quadrant alignment          */
   waypoint_angle=normalize(degrees(atan2 (y_command - y, x_command - x)));
```

```
        psi_command     = waypoint_angle;

        if (TRACE)
        {
            printf ("WAYPOINTCONTROL psi_command = %5.1f, ", psi_command);
            printf ("x = %5.1f, y = %5.1f\n", x, y);
        }
        if ((FOLLOWWAYPOINTMODE == TRUE) && (HOVERCONTROL == FALSE) &&
            ((    waypoint_distance > standoff_distance) ||
             (fabs ( z - z_command) > standoff_distance)))
        {
            if (TRACE) printf ("[FOLLOWWAYPOINTMODE check]");
            /* continue until WAYPOINT reached without further script orders */
            time_next_command = t + 2.0 * dt;
        }
        else /* WAYPOINT reached */
        {
          if (TRACE) printf ("[FOLLOWWAYPOINTMODE success, WAYPOINT reached]");
              fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                               t, psi_command, x_command, y_command, z_command,
                             port_rpm_command, stbd_rpm_command,
                               rudder_command,   planes_command,
                                   bow_lateral_thruster_command,
                                 stern_lateral_thruster_command,
                                  bow_vertical_thruster_command,
                                stern_vertical_thruster_command);
        }
    }
    else if (HOVERCONTROL == TRUE)
    {
       waypoint_angle=normalize(degrees(atan2(y_command - y, x_command - x)));

       track_angle = normalize (waypoint_angle - psi);
       along_track_distance =   cos (radians(track_angle)) * waypoint_distance;
       cross_track_distance = - sin (radians(track_angle)) * waypoint_distance;

       port_rpm = k_propeller_hover * along_track_distance -
                       k_surge_hover * u;
       stbd_rpm = k_propeller_hover * along_track_distance -
                       k_surge_hover * u;

       if (TRUE)
       {
           printf ("HOVERCONTROL:\n");
           printf ("psi_command           = %5.1f, ", psi_command);
           printf ("x = %5.1f, y = %5.1f\n", x, y);
           printf ("waypoint_distance    = %5.1f, track_angle = %5.1f\n",
                    waypoint_distance,          track_angle);
           printf ("along_track_distance = %5.1f, ", along_track_distance);
           printf ("cross_track_distance = %5.1f\n", cross_track_distance);
           printf ("port_rpm / stbd_rpm  = %5.1f\n", port_rpm);
       }
    }

    /* Simplified PD rudders/planes control rules: - - - - - - - - - - - - - */

    delta_rudder =    k_psi * normalize2 (psi - psi_command)

                   + (k_r  * r) + (k_v  * v);

    /*  tanh not provided under OS-9 C, added at end of this program         */

    if (SLIDINGMODECOURSE == TRUE)
    {
       sigma = k_sigma_r * r + k_sigma_psi * normalize2 (psi - psi_command);
```

```
    delta_rudder =  (3.1403 * r) + 81.9712 * eta_steering * tanh (sigma);
 }

 depth_error  = (z - z_command);

 /* constrain depth_error to +- 15.0 feet to prevent going vertical        */
 /*       and enable stable pitch angle even on large depth changes        */

 clamp (& depth_error, -15.0, 15.0, "depth_error");               /* feet */

 delta_planes = - (k_z * depth_error)

                + (k_theta * theta) + (k_q * q) - (k_w * w);

 /* Dead stick means no open loop control of rudders/planes - - - - - - - */

 if (DEADSTICKRUDDER)
 {
     delta_rudder = rudder_command;
 }
 if (DEADSTICKPLANES)
 {
     delta_planes = planes_command;
 }

 /* constrain planes & rudder orders +/- 40.0 degrees                      */

 clamp (& delta_rudder, -40.0, 40.0, "delta_rudder");          /* degrees */
 clamp (& delta_planes, -40.0, 40.0, "delta_planes");          /* degrees */

 /* Send commands to rudders and planes *********************************/

 rudder (delta_rudder);
 planes (delta_planes);

 /* Simplified lateral/vertical thruster control rules: - - - - - - - - - */

 if       (ROTATECONTROL == TRUE)  /* open loop rotate  thrusters */
 {
    lateral_thruster_volts =   k_thruster_rotate * rotate_command
                                                 * rotate_command;
 }
 else if (LATERALCONTROL == TRUE) /* open loop lateral thrusters */
 {
    lateral_thruster_volts = - k_thruster_lateral * (lateral_command);
 }
 else                             /* heading control is default */
 {
    lateral_thruster_volts = - k_thruster_psi * normalize2 (psi-psi_command)

                             - k_thruster_r   * r;
 }

 vertical_thruster_volts   = - k_thruster_z * (z - z_command)

                             - k_thruster_w * w;

 if ((THRUSTERCONTROL) || (HOVERCONTROL == TRUE) || (ROTATECONTROL == TRUE))
 {
     AUV_bow_vertical   =   vertical_thruster_volts;
     AUV_stern_vertical =   vertical_thruster_volts;

     if       (LATERALCONTROL == TRUE)
     {
        AUV_bow_lateral    =    lateral_thruster_volts; /* both positive, */
        AUV_stern_lateral  =    lateral_thruster_volts; /* same direction */
     }
```

```
    else if (HOVERCONTROL == TRUE)
    {
      AUV_bow_lateral     = - ( - k_thruster_psi*normalize2(psi-psi_command)
                                - k_thruster_r   * r)
                            + k_thruster_hover  * cross_track_distance
                            + k_sway_hover      * v;
      AUV_stern_lateral  =    ( - k_thruster_psi*normalize2(psi-psi_command)
                                - k_thruster_r   * r)
                            + k_thruster_hover  * cross_track_distance
                            + k_sway_hover      * v;
    }
    else if ((THRUSTERCONTROL == TRUE) || (ROTATECONTROL == TRUE))
    {
      AUV_bow_lateral    =  - lateral_thruster_volts; /* negative */
      AUV_stern_lateral  =    lateral_thruster_volts;
    }
    else
    {
      printf ("Thruster control logic error *** \n");
    }

    if (TRACE && DISPLAYSCREEN)
    {
       printf ( "Thruster control ON.  Pre-clamp calculated values:\n");
       printf (" AUV_bow_vertical = %6.3f, AUV_stern_vertical = %6.3f\n",
                 AUV_bow_vertical,          AUV_stern_vertical);
       printf (" AUV_bow_lateral  = %6.3f, AUV_stern_lateral  = %6.3f\n",
                 AUV_bow_lateral,           AUV_stern_lateral);
    }
}
else /* thrusters disabled */
{
    if (TRACE && DISPLAYSCREEN)
    {
       printf ( "Thruster control OFF.  Pre-clamp calculated values:\n");
       printf ("vertical_thruster_volts = %6.3f\n",
               vertical_thruster_volts);
       printf ( "lateral_thruster_volts = %6.3f\n",
                lateral_thruster_volts);
    }
    AUV_bow_vertical   =  0.0;
    AUV_stern_vertical =  0.0;
    AUV_bow_lateral    =  0.0;
    AUV_stern_lateral  =  0.0;
}

if (TRACE && DISPLAYSCREEN)
{
    printf ("Pre-sqrt thruster control calculated values:\n");
    printf ("AUV_bow_vertical   = %6.3f\n", AUV_bow_vertical);
    printf ("AUV_stern_vertical = %6.3f\n", AUV_stern_vertical);
    printf ("AUV_bow_lateral    = %6.3f\n", AUV_bow_lateral);
    printf ("AUV_stern_lateral  = %6.3f\n", AUV_stern_lateral);
}

/* convert to signed sqrt to account for volts-to-thrust relationship   */

AUV_bow_vertical   = 2.0 * sqrt(6.0) * sign (AUV_bow_vertical   )
                            * sqrt (fabs (AUV_bow_vertical   ));
AUV_stern_vertical = 2.0 * sqrt(6.0) * sign (AUV_stern_vertical)
```

```
                                        * sqrt (fabs (AUV_stern_vertical));
AUV_bow_lateral     = 2.0 * sqrt(6.0) * sign (AUV_bow_lateral    )
                                        * sqrt (fabs (AUV_bow_lateral    ));
AUV_stern_lateral   = 2.0 * sqrt(6.0) * sign (AUV_stern_lateral )
                                        * sqrt (fabs (AUV_stern_lateral ));

if (TRACE && DISPLAYSCREEN)
{
    printf ("Post-sqrt thruster control calculated values:\n");
    printf ("AUV_bow_vertical   = %6.3f\n", AUV_bow_vertical);
    printf ("AUV_stern_vertical = %6.3f\n", AUV_stern_vertical);
    printf ("AUV_bow_lateral    = %6.3f\n", AUV_bow_lateral);
    printf ("AUV_stern_lateral  = %6.3f\n", AUV_stern_lateral);
}

/* constrain thruster orders +/- 24.0 volts == 3820 rpm no-load              */

clamp (&  AUV_bow_vertical,   -24.0, 24.0, "AUV_bow_vertical");
clamp (&  AUV_stern_vertical, -24.0, 24.0, "AUV_stern_vertical");
clamp (&  AUV_bow_lateral,    -24.0, 24.0, "AUV_bow_lateral");
clamp (&  AUV_stern_lateral,  -24.0, 24.0, "AUV_stern_lateral");

if ((port_rpm_command == 0.0) && (stbd_rpm_command == 0.0))
/* prevent planes chatter by zeroing them at zero speed                      */
{
    delta_rudder =  0.0;
    delta_planes =  0.0;
}

/* Normalization within bounds - - - - - - - - - - - - - - - - - - - - - - - */

delta_rudder =   normalize2 (delta_rudder);
delta_planes =   normalize2 (delta_planes);

/* constrain planes & rudder orders +- 40.0 degrees                          */
if (fabs (delta_rudder) > (40.0 /* degrees */))
{
          delta_rudder  = (40.0) * (delta_rudder / fabs (delta_rudder));
}
if (fabs (delta_planes) > (40.0 /* degrees */))
{
          delta_planes  = (40.0) * (delta_planes / fabs (delta_planes));
}

/* Send commands to rudders and planes **********************************/

rudder (delta_rudder);
planes (delta_planes);

/* Send command & get reply from sonar **********************************/

AUV_ST1000_bearing = 0.0;           /* relative bearings of sonar heads */
AUV_ST725_bearing  = 0.0;

/* send telemetry to tactical level and data recording files - - - - - - */
record_data ();

/* read commands from tactical level - - - - - - - - - - - - - - - - - - */
if (TACTICAL == TRUE) read_parallel_port ();


/* update simulation clock "t" - - - - - - - - - - - - - - - - - - - - - */

t = t + dt;

fflush (stdout);
currentloopclock = clock ();
```

```
    if ((REALTIME == TRUE) && LOCATIONLAB &&
        (currentloopclock < previousloopclock
                            + (int)(dt * (float) CLOCKS_PER_SEC)))
    {
         if (TRACE && DISPLAYSCREEN)
         {
            printf ("currentloopclock = %ld, previousloopclock = %ld\n",
                     currentloopclock,      previousloopclock);
       printf("timestep dt = %5.3f seconds (corresponding clock ticks = %d)\n",
                     dt, (int)(dt * (float) CLOCKS_PER_SEC));
            printf ("Busy wait until system clock reaches simulation clock, ");
            printf ("loop duration = %5.3f\n",
                     ((float) currentloopclock - (float) previousloopclock)
                      / CLOCKS_PER_SEC);
         }
         while  (currentloopclock < previousloopclock
                                    + (int)(dt * (float) CLOCKS_PER_SEC))
         {
             currentloopclock = clock (); /* %%%%% busy wait %%%%% */
         }
         if (TRACE && DISPLAYSCREEN)
         {
             printf ("Busy wait complete, loop+wait duration = %5.3f, ",
                     ((float) currentloopclock - (float) previousloopclock)
                      / CLOCKS_PER_SEC);
             printf ("current clock () = %ld\n", currentloopclock);
         }
    }
    else if ((REALTIME == FALSE) && LOCATIONLAB && DISPLAYSCREEN && TRACE)
         {
             printf ("No busy wait, loop duration = %5.3f, ",
                     ( (float) currentloopclock - (float) previousloopclock)
                     / CLOCKS_PER_SEC);
             printf ("current clock () = %ld\n", currentloopclock);
         }
    previousloopclock = clock ();

    /* estimate X and Y by dead reckoning  - - - - - - - - - - - - - - - - - - */

    x = x + (speed * dt * cos (radians (psi)));
    y = y + (speed * dt * sin (radians (psi)));

    if   (feof (auvscriptfile) && (t >= time_next_command)) /* all done */
    {
         if (TRUE && DISPLAYSCREEN) printf ("end_test set TRUE\n");
         end_test = TRUE;
    }
    else if (t >= time_next_command) /* scriptfile not yet closed, read more */
    {
         if (TRUE && DISPLAYSCREEN)
            printf ("\n[read more from parse_mission_script_commands]\n");
         parse_mission_script_commands ();    /* get next script orders read */
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish closed_loop_control_module ()]\n");

    return;

} /* end closed_loop_control_module () */

/*****************************************************************************/

void get_control_constants ()
                                      /* get data from file at program start */
{
    if (TRACE && DISPLAYSCREEN)
```

```
    printf ("[start get_control_constants ()]\n");

  if (ENTERCONTROLCONSTANTS)      /* - - - - - - - - - - - - - - - - - - - - - - - - - */
  {
    printf("Input start_dwell\n");
    scanf("%d", &start_dwell);

    /* note %F required by OS-9, accepted by SGI */
    printf("Input  k_psi, k_r, k_v\n");
    scanf("%F %F %F", &k_psi, &k_r, &k_v);

    printf("Input k_z, k_w, k_theta, and k_q\n");
    scanf("%F %F %F %F", &k_z, &k_w, &k_theta, &k_q);

    printf("Input k_thruster_psi,k_thruster_r\n");
    scanf("%F %F", &k_thruster_psi, &k_thruster_r);

    printf("Input k_thruster_rotate\n");
    scanf("%F", &k_thruster_rotate);

    printf("Input k_thruster_z,k_thruster_w\n");
    scanf("%F %F", &k_thruster_z, &k_thruster_w);

    printf("Input k_propeller_hover, k_surge_hover\n");
    scanf("%F %F", &k_propeller_hover, &k_surge_hover);

    printf("Input k_thruster_hover, k_sway_hover\n");
    scanf("%F %F", &k_thruster_hover,
                   &k_sway_hover);

    printf("Input speed_limit from 1 to 2 feet/sec \n");
    scanf("%F",&speed_limit);

    printf("Input rpm from +-200.0 to +-700.0\n");
    scanf("%F",&rpm);
  }
  else /* use default initialization values - - - - - - - - - - - - - - - - */
  {
    if (TRACE && DISPLAYSCREEN)
        printf ("[using default control constant values]\n");

    start_dwell =   1;      /* delay time in seconds */

    k_psi        =    1.00; /* degrees rudder per degree of course error */
    k_r          =    2.00; /* degrees rudder per degree/sec yaw rate    */
    k_v          =    0.00; /* needed ?? */

    k_z          =   15.0;  /* degrees planes per foot of depth error    */
    k_w          =    2.0;
    k_theta      =    4.0;
    k_q          =    1.0;

    rpm                = 400.0;

    k_thruster_psi     =    0.6; /* volts per 1 degree course error       */
    k_thruster_r       =    5.0;
    k_thruster_rotate  =   2.25; /* (24V)^2=> 2 # = 16.0 deg/sec empirical*/
                                 /* k_thruster_rotate=(24V / 16 deg/sec)^2*/
    k_thruster_lateral=   48.0; /* 24 V = 2 # =  0.5  ft/sec empirically */
                                 /* note voltage follows a square law     */
    k_thruster_z       =   50.0; /* guesses */
    k_thruster_w       =   50.0;

    k_propeller_hover =  200.0; /*  200 rpm per one  foot error           */

    k_surge_hover      = 6000.0; /*   60 rpm per 0.01 foot/sec surge      */
```

```
                                       /* this value is high to reduce sternway */

    k_thruster_hover  =    4.0;
    k_sway_hover      =   40.0;

}
speed_limit =     2.0;         /* 1.0 to 2.0 ft/sec */

if (TRACE && DISPLAYSCREEN)
{
    printf ("[k_psi = %5.2f, k_r = %5.2f, k_v = %5.2f, k_z = %5.2f, ",
             k_psi,       k_r,         k_v,         k_z);
    printf ("k_w = %5.2f, k_theta = %5.2f, k_q = %5.2f]\n",
            k_w,          k_theta,         k_q);
    printf ("[k_thruster_psi = %5.2f, k_thruster_r = %5.2f, ",
             k_thruster_psi,          k_thruster_r);
    printf ("[k_thruster_rotate = %5.2f, ",
             k_thruster_rotate);
    printf ("k_thruster_z = %5.2f, k_thruster_w = %5.2f]\n",
            k_thruster_z,          k_thruster_w);

    printf ("k_propeller_hover = %5.2f, k_surge_hover = %5.2f]\n",
            k_propeller_hover,          k_surge_hover);

    printf ("k_thruster_hover = %5.2f,  k_sway_hover = %5.2f]\n",
            k_thruster_hover,           k_sway_hover);
}
if ((controlconstantsfile = fopen (CONTROLCONSTANTSFILENAME,"w")) == NULL)
{
    printf("AUV execution:  unable to open output file %s for writing.  ",
           CONTROLCONSTANTSFILENAME);
    printf
    ("                Check ownership permissions in current directory.\n");
    printf("Exit.\n");
    exit  (-1);
}
if (TRACE && DISPLAYSCREEN)
    printf ("[controlconstantsfile %s open, pointer = %x]\n",
             CONTROLCONSTANTSFILENAME, controlconstantsfile);

fprintf (controlconstantsfile,
         " _____________________________________________________ \n\n");
fprintf (controlconstantsfile,
         " AUV execution level control algorithm coefficients \n");
fprintf (controlconstantsfile,
         " _____________________________________________________ \n\n\n");
fprintf (controlconstantsfile,
         " k_psi    k_r    k_v    k_z    k_w    k_theta     k_q \n\n");
fprintf (controlconstantsfile,
         " %5.2f   %5.2f  %5.2f  %5.2f  %5.2f   %5.2f      %5.2f \n\n\n\n",
          k_psi,    k_r,   k_v,   k_z,   k_w,   k_theta,    k_q );

fprintf (controlconstantsfile,
         " k_thruster_psi      k_thruster_r       k_thruster_rotate\n\n");
fprintf (controlconstantsfile,
         "      %5.2f               %5.2f                      %5.2f \n\n\n\n",
          k_thruster_psi,     k_thruster_r,      k_thruster_rotate);

fprintf (controlconstantsfile,
         " k_thruster_z        k_thruster_w \n\n");
fprintf (controlconstantsfile,
         "      %5.2f               %5.2f     \n\n\n\n",
          k_thruster_z,       k_thruster_w);

fprintf (controlconstantsfile,
         " k_propeller_hover   k_surge_hover \n\n");
```

```
    fprintf (controlconstantsfile,
                "       %5.2f              %5.2f    \n\n\n\n",
                    k_propeller_hover,  k_surge_hover);

    fprintf (controlconstantsfile,
                "  k_thruster_hover      k_sway_hover \n\n");
    fprintf (controlconstantsfile,
                "     %5.2f              %5.2f    \n\n\n\n",
                    k_thruster_hover,   k_sway_hover);

    fflush (controlconstantsfile);
    fclose (controlconstantsfile);

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish get_control_constants ()]\n");

    return;

} /* end get_control_constants () */

/*************************************************************************************/

double depth ()
{
     int    val       = 0;
     double new_z     = 0.0;
     double z_offset = 0.0;

     if (TRACE && DISPLAYSCREEN) printf ("[start  depth ()]\n");

     if (LOCATIONLAB && DEADRECKON)
     {
         z = z_command;
     }

if (NOT_YET_REIMPLEMENTED)
{
     z_offset = 0.0;
     val      = adc2(0,0);
     new_z    = 0.002237 * (val - z_val0) + z_offset;     /* new_z (ft) */
}
else new_z = z;

     if (TRACE && DISPLAYSCREEN)
         printf ("[finish depth (), returns %5.3f]\n", new_z);

     return (new_z);

} /* end depth () */

/*************************************************************************************/

double calculate_psi ()
{

     unsigned short psi_bit;
     int psi_bit_int,psi_bit_old_int,delta_psi_bit;
     double angle,tpi;
     double pi = 3.1415927;

     if (TRACE && DISPLAYSCREEN) printf ("[start  calculate_psi ()]\n");

     if (LOCATIONLAB && DEADRECKON)
     {
         psi = psi_command;
     }
```

```
if (NOT_YET_REIMPLEMENTED)
{

      /* port needs to be redone:  */
      psi_bit = Read_PortAB((struct MFI_PIA *) MFI_BASE);

      psi_bit &= 0x3FFF;
      psi_bit_int = psi_bit;
      psi_bit_old_int = psi_bit_old;

      delta_psi_bit = psi_bit_int - psi_bit_old_int;
      psi_bit_old = psi_bit;

      if(abs(delta_psi_bit) > 10000)
      {
         wrap_count = wrap_count - delta_psi_bit/abs(delta_psi_bit);
      }
      tpi = 2.0 * pi * wrap_count;

      angle = heading() - dg_offset + tpi;

      angle = degrees (angle);

} else angle = psi;

  if (TRACE && DISPLAYSCREEN)
      printf ("[finish calculate_psi () returns %5.3f]\n", angle);

  return (normalize (angle));

}  /* end calculate_psi () */

/***************************************************************************/

void zero_gyro_data ()
{
     int index;
     int save_trace;

     save_trace = TRACE;

     if (TRACE && DISPLAYSCREEN) printf ("[start  zero_gyro_data ()]\n");

     pitch_0       = adc1(6);
     roll_0        = adc1(7);
     roll_rate_0   = adc1(9);
     pitch_rate_0  = adc1(8);
     yaw_rate_0    = adc1(10);
     z_val0        = adc2(0,0);
     dg_offset     = heading();

     if (TRACE && DISPLAYSCREEN)
                              printf ("[device averaging for 2 seconds... ]\n");
     for (index=0;index<99;++index)
     {
        pitch_0       += adc1(6);
        roll_0        += adc1(7);
        roll_rate_0   += adc1(9);
        pitch_rate_0  += adc1(8);
        yaw_rate_0    += adc1(10);
        z_val0        += adc2(0,0);

        TRACE          = FALSE;
        dg_offset     += heading(); /* this is verbose if TRACEd */
        TRACE          = save_trace;
        tsleep (5);                 /* 256ths of a second       */
```

```
    }

    pitch_0       = pitch_0/100;
    roll_0        = roll_0/100;
    roll_rate_0   = roll_rate_0/100;
    pitch_rate_0  = pitch_rate_0/100;
    yaw_rate_0    = yaw_rate_0/100;
    z_val0        = z_val0/100;
    dg_offset     = dg_offset/100.0;

    if (TRACE && DISPLAYSCREEN)
    {
       printf ("pitch_0      = %d\n", pitch_0);
       printf ("roll_0       = %d\n", roll_0);
       printf ("roll_rate_0  = %d\n", roll_rate_0);
       printf ("pitch_rate_0 = %d\n", pitch_rate_0);
       printf ("yaw_rate_0   = %d\n", yaw_rate_0);
       printf ("z_val0       = %d\n", z_val0);
       printf ("dg_offset    = %f\n", dg_offset);
    }

    if (TRACE && DISPLAYSCREEN) printf ("[finish zero_gyro_data ()]\n");

    return;

}  /* end zero_gyro_data () */

/*****************************************************************************/

void initialize_dacs ()                /* Initialize all dac channels to zero */
{
    if (TRACE && DISPLAYSCREEN) printf ("[start  initialize_dacs ()]\n");

if (NOT_YET_REIMPLEMENTED)
{
    control_surface (FRONT_RUD_TOP,0.0);
    control_surface (FRONT_RUD_BOT,0.0);
    control_surface (FRONT_PL_RIGHT,0.0);
    control_surface (FRONT_PL_LEFT,0.0);
    control_surface (REAR_RUD_TOP,0.0);
    control_surface (REAR_RUD_BOT,0.0);
    control_surface (REAR_PL_RIGHT,0.0);
    control_surface (REAR_PL_LEFT,0.0);

    main_motors_off ();

}
    if (TRACE && DISPLAYSCREEN) printf ("[finish initialize_dacs ()]\n");

    return;

} /* end initialize_dacs */


/*****************************************************************************/

void initialize_adcs ()
{
    if (TRACE && DISPLAYSCREEN) printf ("[start  initialize_adcs ()]\n");

#if defined (sgi)
#else
    /* Initialize MFI channels:  0 = input port, 1 = output port */

    Init_PortA ((struct MFI_PIA *) MFI_BASE, MFI_INPUT_PORT);
    Init_PortB ((struct MFI_PIA *) MFI_BASE, MFI_INPUT_PORT);
#endif
```

```
    if (TRACE && DISPLAYSCREEN) printf ("[finish initialize_adcs ()]\n");

    return;

} /* end initialize_adcs */

/*****************************************************************************/

void control_surface (surface, angle)

  /* This function sends the desired ANGLE to the specified control SURFACE
     The angle is first normalized to (-45 to 45), then correction is applied
     for the non-linearity in the servo control module */

    int     surface;

    double angle;
{
    int volt;
    double a,b,c,d;

    if (FALSE && DISPLAYSCREEN) printf ("[start  control_surface ()]\n");

if (NOT_YET_REIMPLEMENTED)
{

    a = 1.2487e-4;
    b = -2.9087e-2;
    c = 5.0927;
    d = 500.6576;

    angle = angle*57.295779; /* Convert RADIANS to DEGREES */

    if ((angle < -22.92) || (angle > 22.92))
    {
       /* Plane saturated set to +- 45 */
       angle = 22.92*angle/fabs(angle);

    }

    volt = a*pow(angle,3.) + b*pow(angle,2.) + c*angle + d;

    dac2b(volt,surface);

}
    if (FALSE && DISPLAYSCREEN) printf ("[finish control_surface ()]\n");

    return;

} /* control_surface */

/*****************************************************************************/

void rudder (angle)

  /* Send angular deflection (RADIANS) to rudders.
     Convention (+) angle left turn, (-) angle right turn */

    double angle;
{
    if (TRACE && DISPLAYSCREEN) printf ("[start  rudder ()]\n");

    control_surface (FRONT_RUD_TOP,-angle);
    control_surface (FRONT_RUD_BOT,angle);
    control_surface (REAR_RUD_TOP,angle);
    control_surface (REAR_RUD_BOT,-angle);
```

```
    if (TRACE && DISPLAYSCREEN) printf ("[finish rudder ()]\n");

    return;

} /* rudder */

/*****************************************************************************/

void planes (angle)

  /* Send angular deflection (RADIANS) to bow and stern planes.
     Convention (+) angle dive, (-) angle rise */

     double angle;
{
     if (TRACE && DISPLAYSCREEN) printf ("[start  planes ()]\n");

     control_surface (FRONT_PL_RIGHT,angle);
     control_surface (FRONT_PL_LEFT,-angle);
     control_surface (REAR_PL_RIGHT,-angle);
     control_surface (REAR_PL_LEFT,  angle);

     if (TRACE && DISPLAYSCREEN) printf ("[finish planes ()]\n");

     return;

} /* end planes () */

/*****************************************************************************/

void main_motors_off ()                          /* Turn off both main motors */
{
     if (TRACE && DISPLAYSCREEN) printf ("[start  main_motors_off ()]\n");

if (NOT_YET_REIMPLEMENTED)
{
     dac1(512,SUPPLY);
     dac1(512,RIGHT_MOTOR);
     dac1(512,LEFT_MOTOR);
}
     if (TRACE && DISPLAYSCREEN) printf ("[finish main_motors_off ()]\n");

     return;

} /* end main_motors_off () */

/*****************************************************************************/

void alive (interval, local_start_dwell)

     unsigned int interval;
              int local_start_dwell;
{
     unsigned int iinterval,jinterval;
     double test_delta;

     if (TRACE && DISPLAYSCREEN) printf ("[start  alive ()]\n");

if (NOT_YET_REIMPLEMENTED)
{

     local_start_dwell = local_start_dwell*100;
     interval = interval*100;
     iinterval = local_start_dwell/interval;
     jinterval = 0;
     test_delta = .4; /* Deflect 22.5 degrees */
```

```
      while(jinterval < iinterval)
      {
         control_surface (FRONT_RUD_TOP,test_delta);
         tsleep(interval);     /* 256ths of a second */
         test_delta = -test_delta;
         jinterval = jinterval + 1;
      }

      tsleep(200);               /* 256ths of a second */
   }
      if (TRACE && DISPLAYSCREEN) printf ("[finish alive ()]\n");

      return;
}

/*****************************************************************************/

void record_data_on ()
{
      if (TRACE && DISPLAYSCREEN) printf ("[start  record_data_on ()]\n");

      /* Open files for writing                                              */

      if ((auvdatafile = fopen (AUVDATAFILENAME,"w")) == NULL)
      {
          printf("AUV execution:  unable to open output file %s for writing.  ",
                 AUVDATAFILENAME);
          printf
         ("                Check ownership permissions in current directory.\n");
          printf("Exit.\n");
          exit  (-1);
      }
      if (TRACE && DISPLAYSCREEN)
          printf ("[auvdatafile %s open,   pointer = %x]\n",
                                 AUVDATAFILENAME, auvdatafile);

      if ((auvtextfile = fopen (AUVTEXTFILENAME,"w")) == NULL)
      {
         printf("AUV execution:  unable to open output file %s for writing.  ",
                AUVTEXTFILENAME);
         printf
         ("                Check ownership permissions in current directory.\n");
         printf("Exit.\n");
         exit  (-1);
      }
      if (TRACE && DISPLAYSCREEN)
          printf ("[auvtextfile %s open, pointer = %x]\n",
                                 AUVTEXTFILENAME, auvtextfile);

      fprintf (auvtextfile, "# auvtextfile %s shows state ", AUVTEXTFILENAME);
      fprintf (auvtextfile, "vector variables at one second intervals.\n\n");

      if (LOOPFOREVER)
          fprintf (auvtextfile, "# Mission replication #%d\n",
                                  replication_count);

/* testing code from wr2t1.c, not currently in use         */
/* serial.d is a telemetry test file to check connectivity */
/*    if ((serialtestfile = fopen ("serial.d", "r")) <= 0)
      {
       printf("AUV execution:  record_data_on: can't open test file serial.d\n");
       printf("Exit.\n");
       exit  (-1);
      }
*/
      if (TRACE && DISPLAYSCREEN) printf ("[finish record_data_on ()]\n");
```

```
    return;
}

/****************************************************************************/

void record_data_off ()

{
    if  (TRACE && DISPLAYSCREEN) printf ("[start  record_data_off ()]\n");

    if  (TRACE && DISPLAYSCREEN)
    {
         printf ("[flushing and closing auvdatafile %s   %x]\n",
                                                AUVDATAFILENAME, auvdatafile);
         fflush (stdout);
    }
    if  (auvdatafile != NULL)
    {
         if (TRACE && DISPLAYSCREEN) printf ("[auvdatafile flushed]\n");
         fflush (stdout);
         fflush (auvdatafile);
         fclose (auvdatafile);
         if (TRACE && DISPLAYSCREEN) printf ("[auvdatafile closed]\n");
         fflush (stdout);
    }
    else if (TRACE && DISPLAYSCREEN) printf ("[auvdatafile was not open!!]\n");

    if  (TRACE && DISPLAYSCREEN)
    {
         printf ("[flushing and closing auvtextfile %s %x]\n",
                                                AUVTEXTFILENAME, auvtextfile);
         fflush (stdout);
    }
    if  (auvtextfile != NULL)
    {
         if (TRACE && DISPLAYSCREEN) printf ("[auvtextfile flushed]\n");
         fflush (stdout);
         fflush (auvtextfile);
         fclose (auvtextfile);
         if (TRACE && DISPLAYSCREEN) printf ("[auvtextfile closed]\n");
         fflush (stdout);
    }
    else if (TRACE && DISPLAYSCREEN) printf ("[auvtextfile was not open!!]\n");

/*  fclose (serialtestfile);  /* serial port test file */

    if (TRACE && DISPLAYSCREEN)
    {
        printf ("[finish record_data_off ()]\n");
        fflush (stdout);
    }
    return;
}

/****************************************************************************/

void record_data ()

{
    /* temporary hold variables */

    double  AUV_time_temp,
            AUV_x_temp,               AUV_y_temp,             AUV_z_temp,
            AUV_phi_temp,             AUV_theta_temp,         AUV_psi_temp,
            AUV_u_temp,               AUV_v_temp,             AUV_w_temp,
            AUV_p_temp,               AUV_q_temp,             AUV_r_temp,
            AUV_x_dot_temp,           AUV_y_dot_temp,         AUV_z_dot_temp,
```

```
        AUV_phi_dot_temp,          AUV_theta_dot_temp,    AUV_psi_dot_temp,
        AUV_delta_rudder_temp,     AUV_delta_planes_temp,
        AUV_port_rpm_temp,         AUV_stbd_rpm_temp,
        AUV_bow_vertical_temp,     AUV_stern_vertical_temp,
        AUV_bow_lateral_temp,      AUV_stern_lateral_temp,
        AUV_ST1000_range_temp,     AUV_ST725_range_temp,
        AUV_ST1000_bearing_temp,   AUV_ST725_bearing_temp,
        AUV_ST1000_strength_temp,AUV_ST725_strength_temp;

  if (TRACE && DISPLAYSCREEN) printf ("[start  record_data ()]\n");

  system_time             = time (NULL);
  system_tmp              = localtime (&system_time);

  if (TRACE && DISPLAYSCREEN)
     printf ("[OS-9 system time is %02d:%02d:%02d, replication %d]\n",
             system_tmp->tm_hour, system_tmp->tm_min, system_tmp->tm_sec,
             replication_count);

  for (i = 0; i <= 255; i++)
  {
      buffer [i] = '\0';   /* zero buffer */
  }

  buffer_size = sprintf (buffer,
 " auv_state %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f
%5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f
%5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f \n",
          t,x,y,z,
          normalize2 (phi   ),
          normalize2 (theta),
          normalize  (psi   ),
          u, v, w,
          normalize2 (p),
          normalize2 (q),
          normalize2 (r),
          x_dot, y_dot, z_dot,
          normalize2 (phi_dot   ),
          normalize2 (theta_dot),
          normalize2 (psi_dot   ),
          normalize2 (delta_rudder),
          normalize2 (delta_planes),
          port_rpm,              stbd_rpm,
          AUV_bow_vertical,      AUV_stern_vertical,
          AUV_bow_lateral,       AUV_stern_lateral,
          AUV_ST1000_bearing,    AUV_ST1000_range,     AUV_ST1000_strength,
          AUV_ST725_bearing,     AUV_ST725_range,      AUV_ST725_strength);

  if (buffer_size > 254)          /* sprintf buffer overflow condition      */
  {
     if (DISPLAYSCREEN)
        printf ("Buffer overflow, buffer_size = %d, reduced to 254 !!!!!!\n",
                                buffer_size);
     buffer_size   =      254;
  }

  if (TRACE && DISPLAYSCREEN) printf ("[buffer_size is %d]\n", buffer_size);

  /* other state variables & timing constraints   <<<<<<<<<<<<<<<<<<<<<<<<<<< */

  send_data_on_virtual_world_socket (); /* there it goes                    */

  if (TRUE && DISPLAYSCREEN)            /* telemetry report to screen       */
  {
     printf ("\nsending to virtual world:");
```

```
        printf ("\n%s", buffer);
    }
    else if (DISPLAYSCREEN)                     /* partial telemetry report         */
    {
        printf ("\nsent telemetry to virtual world %5.3f  ", t);
    }

    get_stream_from_virtual_world_socket ();     /* here it comes */

    /* update and output AUV state variables                                       */
    /* note that if dead reckoning is used, values will not change                 */

    variables_parsed = sscanf(buffer_received,
/*          "%s %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf
%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf \n",*/
            "%s %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F %F
%F %F %F %F %F %F %F %F %F %F \n",
             keyword,                    &AUV_time_temp,
            &AUV_x_temp,                 &AUV_y_temp,                &AUV_z_temp,
            &AUV_phi_temp,               &AUV_theta_temp,            &AUV_psi_temp,
            &AUV_u_temp,                 &AUV_v_temp,                &AUV_w_temp,
            &AUV_p_temp,                 &AUV_q_temp,                &AUV_r_temp,
            &AUV_x_dot_temp,             &AUV_y_dot_temp,            &AUV_z_dot_temp,
            &AUV_phi_dot_temp,           &AUV_theta_dot_temp,        &AUV_psi_dot_temp,
            &AUV_delta_rudder_temp,      &AUV_delta_planes_temp,
            &AUV_port_rpm_temp,          &AUV_stbd_rpm_temp,
            &AUV_bow_vertical_temp,      &AUV_stern_vertical_temp,
            &AUV_bow_lateral_temp,       &AUV_stern_lateral_temp,
            &AUV_ST1000_bearing_temp,    &AUV_ST1000_range_temp,
                                         &AUV_ST1000_strength_temp,
            &AUV_ST725_bearing_temp,     &AUV_ST725_range_temp,
                                         &AUV_ST725_strength_temp);

    if (variables_parsed == 34)   /* transfer was OK so keep new values */
    {
        t                  =           (AUV_time_temp);
        x                  =           (AUV_x_temp);
        y                  =           (AUV_y_temp);
        z                  =           (AUV_z_temp);
        phi                =           (AUV_phi_temp);
        theta              =           (AUV_theta_temp);
        psi                =           (AUV_psi_temp);
        u                  =           (AUV_u_temp);
        v                  =           (AUV_v_temp);
        w                  =           (AUV_w_temp);
        p                  =           (AUV_p_temp);
        q                  =           (AUV_q_temp);
        r                  =           (AUV_r_temp);
        x_dot              =           (AUV_x_dot_temp);
        y_dot              =           (AUV_y_dot_temp);
        z_dot              =           (AUV_z_dot_temp);
        phi_dot            =           (AUV_phi_dot_temp);
        theta_dot          =           (AUV_theta_dot_temp);
        psi_dot            =           (AUV_psi_dot_temp);
        delta_rudder       =           (AUV_delta_rudder_temp);
        delta_planes       =           (AUV_delta_planes_temp);
        port_rpm           =           (AUV_port_rpm_temp);
        stbd_rpm           =           (AUV_stbd_rpm_temp);
        AUV_bow_vertical   =           (AUV_bow_vertical_temp);
        AUV_stern_vertical =           (AUV_stern_vertical_temp);
        AUV_bow_lateral    =           (AUV_bow_lateral_temp);
        AUV_stern_lateral  =           (AUV_stern_lateral_temp);
        AUV_ST1000_bearing =normalize(AUV_ST1000_bearing_temp);
        AUV_ST1000_range   =           (AUV_ST1000_range_temp);
        AUV_ST1000_strength =          (AUV_ST1000_strength_temp);
        AUV_ST725_bearing  =normalize(AUV_ST725_bearing_temp);
        AUV_ST725_range    =           (AUV_ST725_range_temp);
```

```
    AUV_ST725_strength  =          (AUV_ST725_strength_temp);
  }
  else if ((variables_parsed != 34) && (variables_parsed != -1))
  {
     if (DISPLAYSCREEN)
printf ("\nGarble problem in buffer_received !!! variables parsed = %d\n%s\n",
              variables_parsed, buffer_received);
     answer = getchar (); /* pause */
     TRACE = TRUE;
  }

  if (LOCATIONLAB == 0) speed = u; /* virtual world speed, not flow sensor */

  if (TRACE && DISPLAYSCREEN)
     printf ("\nfrom virtual world buffer_received:\n%s", buffer_received);

  if (TRUE && DISPLAYSCREEN)
  {
     printf ("\nfrom virtual world state variables:");
     printf ("\n %s t=%5.3f x=%5.3f y=%5.3f z=%5.3f ",
         keyword,              t,
         x,                    y,                    z);
     printf ("phi=%5.3f theta=%5.3f psi=%5.3f ",
         phi,                  theta,                psi);
     printf ("u=%5.3f v=%5.3f w=%5.3f p=%5.3f q=%5.3f r=%5.3f ",
         u,                    v,                    w,
         p,                    q,                    r);
     printf ("x_dot=%5.3f y_dot=%5.3f z_dot=%5.3f ",
         x_dot,                y_dot,                z_dot);
     printf ("phi_dot=%5.3f theta_dot=%5.3f psi_dot=%5.3f ",
         phi_dot,              theta_dot,            psi_dot);
     printf ("delta_rudder=%5.3f delta_planes=%5.3f ",
         delta_rudder,         delta_planes);
     printf ("port_rpm=%5.3f stbd_rpm=%5.3f ",
         port_rpm,             stbd_rpm);
     printf ("bow_vertical=%5.3f stern_vertical=%5.3f ",
         AUV_bow_vertical,     AUV_stern_vertical);
     printf ("bow_lateral=%5.3f stern_lateral=%5.3f ",
         AUV_bow_lateral,      AUV_stern_lateral);
     printf ("ST1000 b/r/s %5.3f %5.3f %5.3f, ST725 b/r/s %5.3f %5.3f %5.3f",
         AUV_ST1000_bearing,   AUV_ST1000_range,     AUV_ST1000_strength,
         AUV_ST725_bearing,    AUV_ST725_range,      AUV_ST725_strength);
     printf (", [current time %d %d %d] \n",
         system_tmp->tm_hour, system_tmp->tm_min, system_tmp->tm_sec);
  }

 /* keep all telemetry variables in degrees */
 phi          = normalize2 (phi   );
 theta        = normalize2 (theta);
 psi          = normalize  (psi   );
 phi_dot      = normalize2 (phi_dot);
 theta_dot    = normalize2 (theta_dot);
 psi_dot      = normalize2 (psi_dot);
 p            = normalize2 (p);
 q            = normalize2 (q);
 r            = normalize2 (r);
 delta_rudder = normalize2 (delta_rudder);
 delta_planes = normalize2 (delta_planes);

 if (TRACE && LOCATIONLAB && DISPLAYSCREEN)
 {
   printf ("------------------------------------------------------------\n");
 }

 if (auvdatafile != NULL)          /* output data to telemetry file        */
 {                                 /* note that unmodified stream is saved */
   if (buffer_size == 0)           /* nothing was received, send auv_state */
```

```
                              fprintf (auvdatafile, "%s", buffer);
        else                         /* feedback was received, send uvw_state */
                              fprintf (auvdatafile, "%s", buffer_received);

        if (TRACE && DISPLAYSCREEN)
            printf("[printed to %s telemetry file]\n", AUVDATAFILENAME);
    }

    /* only send/print out every 10th telemetry entry to tactical level        */
    /*    due to serial port bandwidth limitations :-(                         */

/* redo this when clock logic is redone ****************************************/

    if ((int)((t+0.05)*10.0) == (int) (t+0.05) *10)
    {
        if (TACTICAL && TRACE && DISPLAYSCREEN)
            printf ("[sending data to tactical level]\n");

#if defined (sgi)
#else
                          /* writeln (serialpath, buffer, 255);    <<<<<<<<<<<  */
        if (TACTICAL == TRUE) write   (serialpath, buffer, 255);
#endif

        if (TACTICAL && TRACE && DISPLAYSCREEN)
            printf ("[write buffer to tactical level serialpath OK]\n");

        if (auvtextfile != NULL)             /* output data to .auv text file     */
        {
            if (TRACE && DISPLAYSCREEN)
                printf ("[sending data to .auv text file]\n");

            fprintf (auvtextfile, "%s", buffer);
            if (buffer_size != 0) /* feedback was received, also send uvw_state */
               fprintf (auvtextfile, "%s", buffer_received);

            if (TRACE && DISPLAYSCREEN)
                printf ("[fprintf to .auv text file OK]\n");
        }
    }
    telemetry_records_saved ++;

    if   (((buffer_index +1) % FILEBUFFERSIZE) == 0) buffer_index = 0;
    else                                             buffer_index ++;

    /* need to copy buffer to buffer_array if caching telemetry  <<<<<<<<<<<<  */

    if (TRACE && DISPLAYSCREEN) printf ("[buffer_index = %d]\n", buffer_index);

    /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
    /* test code to send data from file serial.d from wr2t1.c                 */
    /* read characters from file, echo characters to screen,                  */
    /*    send characters to serialpath /t1 device                            */
    /*     while ((c[0] = getc(serialtestfile)) != EOF)
           {
               putc    (c[0],stdout);
               writeln (serialpath, c, 1);
           }
    */
    /* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

    if (TRACE && DISPLAYSCREEN) printf ("[finish record_data ()]\n");

    return;
}
```

```
/*****************************************************************************/

double roll_angle ()      /* Return roll angle in RADIANS                    */
{
     int    val;
     double angle = phi;     /* previous (or virtual world) value */

     if (TRACE && DISPLAYSCREEN) printf ("[start  roll_angle ()]\n");

if (NOT_YET_REIMPLEMENTED)
{
     val = adc1(ROLL_ANGLE_CH);
/*   angle = ((516.578 - val)/5.7572)/57.295779;  convert to radians */
     angle = (-.1737*val + .1737*roll_0)/57.295779;
}

  angle = normalize2 (angle);

  if (TRACE && DISPLAYSCREEN)
     printf ("[finish roll_angle () returns %5.3f]\n", angle);

  return (angle);
}

/*****************************************************************************/

double pitch_angle ()     /* Return pitch angle in RADIANS                   */
{
     int    val;
     double angle = theta;    /* previous (or virtual world) value */

     if (TRACE && DISPLAYSCREEN) printf ("[start  pitch_angle ()]\n");

if (NOT_YET_REIMPLEMENTED)
{

     val = adc1(PITCH_ANGLE_CH);
/*   angle = ((520.153 - val)/8.340)/57.295779;  convert to radians */
     angle = -((-.1199*val + .1199*pitch_0)/57.295779);

}

  angle = normalize2 (angle);

  if (TRACE && DISPLAYSCREEN)
     printf ("[finish pitch_angle () returns %5.3f]\n", angle);

  return (angle);
}

/*****************************************************************************/

double heading ()

  /* Return heading angle with respect to local magnetic north in radians
       from       directional gyro */
{
    unsigned short dg_bit;
    double         angle;

    if (TRACE && DISPLAYSCREEN) printf ("[start  heading ()]\n");

    if       (LOCATIONLAB && (DEADRECKON == FALSE))
    {
        angle = psi;
    }
    else if (LOCATIONLAB && (DEADRECKON == TRUE))
```

```
     {
         angle = psi_command;
     }
     else
     {
if (NOT_YET_REIMPLEMENTED)
{
        dg_bit = Read_PortAB(MFI_BASE);
        dg_bit &= 0x3FFF;
}
        angle = (3.8350e-4) * dg_bit;
     }

     angle = normalize (angle);

     if (TRACE && DISPLAYSCREEN)
         printf ("[finish heading () returns %5.3f]\n", angle);

     return (angle);
}

/*********************************************************************************/

double roll_rate_gyro ()     /* Return roll rate in RADIANS/SEC                  */
{
     int    val;
     double rate = p;  /* previous (or virtual world) value */

     if (TRACE && DISPLAYSCREEN) printf ("[start  roll_rate_gyro ()]\n");

#if defined (sgi)
#else
     val  = adc1 (ROLL_RATE_CH);
     rate = (roll_rate_0/3.2113 - .31062*val)/57.295779;
#endif

     rate = normalize2 (rate);

     if (TRACE && DISPLAYSCREEN)
         printf ("[finish roll_rate_gyro () returns %5.3f]\n", rate);

     return (rate);
}

/*********************************************************************************/

double pitch_rate_gyro ()     /* Return pitch rate in RADIANS/SEC                */
{
     int    val  = 0;
     double rate = q;  /* previous (or virtual world) value */

     if (TRACE && DISPLAYSCREEN) printf ("[start  pitch_rate_gyro ()]\n");

#if defined (sgi)
#else
     val   = adc1 (PITCH_RATE_CH);
     rate  = (pitch_rate_0/13.69399 - .0730001*val)/57.295779;
#endif

     rate = normalize2 (rate);

     if (TRACE && DISPLAYSCREEN)
         printf ("[finish pitch_rate_gyro () returns %5.3f]\n", rate);

     return (rate);
}
```

```
/******************************************************************************/

double yaw_rate_gyro ()     /* Return yaw rate in RADIANS/SEC                 */
{
     int    val  = 0;
     double rate = r;  /* previous (or virtual world) value */

     if (TRACE && DISPLAYSCREEN) printf ("[start  yaw_rate_gyro ()]\n");

#if defined (sgi)
#else
     val  = adc1(YAW_RATE_CH);
     rate = (yaw_rate_0/13.653216 - .0732362*val)/57.295779;
#endif

     rate = normalize2 (rate);

     if (TRACE && DISPLAYSCREEN)
         printf ("[finish yaw_rate_gyro () returns %5.3f]\n", rate);

     return (rate);
}

/******************************************************************************/

double stbd_motor_rpm ()     /* Reads rpm from RIGHT_MOTOR                     */
{
      int    pulse;
      double local_stbd_rpm;

     if (TRACE && DISPLAYSCREEN) printf ("[start  stbd_motor_rpm ()]\n");

     pulse     = adc1(RIGHT_MOTOR_RPM);
     local_stbd_rpm = 1.244*pulse - 8.4792;

     if (TRACE && DISPLAYSCREEN)
          printf ("[finish stbd_motor_rpm () returns %5.3f]\n", stbd_rpm);

     return (local_stbd_rpm);
}

/******************************************************************************/

double port_motor_rpm ()     /* Reads rpm from LEFT_MOTOR                      */
{
      int    pulse;
      double local_port_rpm;

     if (TRACE && DISPLAYSCREEN) printf ("[start  port_motor_rpm ()]\n");

     pulse     = adc1(LEFT_MOTOR_RPM);
     local_port_rpm = 1.244*pulse - 8.4792;

     if (TRACE && DISPLAYSCREEN)
         printf ("[finish port_motor_rpm () returns %5.3f]\n", local_port_rpm);

     return (local_port_rpm);
}
/******************************************************************************/

double get_speed ()  /* Filter the speed signal                                */
{
     int    index;
     double avg_speed = 0.0;

     if (TRACE && DISPLAYSCREEN)
         printf ("[start  get_speed (), LOCATIONLAB=%d]\n", LOCATIONLAB);
```

```
    if (LOCATIONLAB)
    {
        if (TRACE && DISPLAYSCREEN)
           printf ("[finish get_speed () returns ");
        avg_speed = (speed_per_rpm * (port_rpm + stbd_rpm) / 2.0);
        if (TRACE && DISPLAYSCREEN)
           printf ("%5.3f]\n", avg_speed);

        return (avg_speed);
    }
    else
    {
        speed_array[pointer] = adc2(1,0);
        for (index=0;index<11;++index)
        {
           avg_speed += speed_array[index];
        }
        avg_speed = avg_speed/11;
        avg_speed = avg_speed * 0.003635;
        if (avg_speed < 0.78)
        {
            if (t < 10.0)
            {
               avg_speed = speed + 0.000045*(rpm-150);
               if (avg_speed > (rpm*0.002)) avg_speed = rpm*0.002;
            }
            else
            {
               avg_speed = 1.0;
            }
            if(t < 2.0) avg_speed = 0.0;
        }
        pointer = pointer + 1;
        if (pointer > 10) pointer = 0;
    }

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish get_speed () returns %5.3f]\n", avg_speed);

    return (avg_speed);
}
/*************************************************************************/

void get_init_avg ()
{
  int index, rng_sum;

  if (TRACE && DISPLAYSCREEN) printf ("[start  get_init_avg ()]\n");

  rng_sum     = 0;
  range_index = 0;

  for(index = 0; index < AVG_PTS; ++index)
  {
    via0[ORB_IRB] = (SONAR_SW1 & SONAR_SW3) | SONAR_TRIG2;
    via0[ORB_IRB] = SONAR_SW1 & SONAR_SW3;
    tsleep(5);
    range = adc2(3,0);

    rng_sum += range;
    range_array[index] = range;
    ++range_index;

  }
  avg_rng = (rng_sum/AVG_PTS) * 1.0;

  if (TRACE && DISPLAYSCREEN) printf ("[finish get_init_avg ()]\n");
```

```
    return;
  }

/******************************************************************************/

void get_avg_rng ()
{
    int index, UPDATE_AVG, int_rng_sum;

    if (TRACE && DISPLAYSCREEN) printf ("[start  get_avg_rng ()]\n");

    UPDATE_AVG  = 0;
    int_rng_sum = 0;

    if (((float)range > avg_rng ) ||
        (fabs((float)range - avg_rng) <= MAX_RNG_DIFF) ||
        (bad_rng >= MAX_BAD_PTS))
          {
            range_array[range_index] = range;
            ++range_index;
            UPDATE_AVG = 1;
            if(bad_rng > MAX_BAD_PTS)
              {
                ++bad_updates;
              }
            if(bad_updates >= MIN_NO_PTS)
              {
                bad_rng = 0;
              }
          }
        else
          {
            ++bad_rng;
          }

  if(UPDATE_AVG)
    {
      for(index = range_index - AVG_PTS; index <= range_index; ++index)
        {
          int_rng_sum += range_array[index];
        }

      avg_rng = int_rng_sum/AVG_PTS * 1.0;
    }
    if (TRACE && DISPLAYSCREEN) printf ("[finish get_avg_rng ()]\n");

    return;
}

/******************************************************************************/
/*
   Hardware control changes *FOLLOWING* hardware upgrade 1993:

   Telemetry to tactical level:       serial port /T1 via driver /TT

   Orders from  tactical level:       parallel port /P via MFI register A

   Sonar:                             interface card device driver /T3

*/
/******************************************************************************/

void open_device_paths ()
{
     if (TRACE && DISPLAYSCREEN) printf ("[start  open_device_paths ()]\n");
#if defined (sgi)
```

```
#else
     /* either /t1 serial port #1 or /tt (high baud rate driver for /t1)      */
     serialpath = open ("/t1", S_IREAD + S_IWRITE);        /* get path number */
     /*                   /tt is device for high baud rate /t1 serial port     */
     if (serialpath <= 0)
     {
        printf ("open_device_paths (): unable to open serialpath /t1.  ");
        printf ("Exit.\n");
        exit (-1);
     }
     if (TRACE && DISPLAYSCREEN)
        printf ("[serialpath /t1 (normal baud rate) open, path number = %d]\n",
                                                                  serialpath);

     if (SONARINSTALLED == TRUE)
     {
        sonarpath  = open ("/t3", S_IREAD + S_IWRITE);      /* get path number */
        /*                   /t3 is device for sonar interface card           */
        if (sonarpath <= 0)
        {
           printf ("open_device_paths (): unable to open sonarpath /t3.  ");
           printf ("Exit.\n");
           exit (-1);
        }
        if (TRACE && DISPLAYSCREEN)
            printf ("[sonarpath  /t3 open, path number = %d]\n", sonarpath);

        tty_mode (sonarpath,1);    /* initialize sonar values */
     }
     else if (TRACE && DISPLAYSCREEN)
             printf ("[sonarpath  /t3 ignored, SONARINSTALLED == FALSE]\n");

     /* other paths:  effectors, depth_sonar, etc. ****************************/

#endif
     if (TRACE && DISPLAYSCREEN) printf ("[finish open_device_paths ()]\n");

     return;
}

/*****************************************************************************/

void close_device_paths ()
{
     if (TRACE && DISPLAYSCREEN) printf ("[start  close_device_paths ()]\n");

     if (serialpath > 0) close (serialpath);  /* test for open before closing */

     else if (TRACE && DISPLAYSCREEN) printf ("[serialpath was not open!]\n");

     if (SONARINSTALLED)
     {
         if (sonarpath > 0) close (sonarpath);

         else if (TRACE && DISPLAYSCREEN) printf("[sonarpath was not open!]\n");
     }

     /* other paths:  effectors, depth_sonar, etc. <<<<<<<<<<<<<<<<<<<<<<<<<< */

     if (TRACE && DISPLAYSCREEN) printf ("[finish close_device_paths ()]\n");

     return;
}
/*****************************************************************************/

void   read_parallel_port ()   /* loop and display 8 bit data from port A */
{
```

```
   static     char  next_char, last_char;
   static     char  current_command [256];
   static     int   index;
   unsigned   char  temp;

   if (TRACE && DISPLAYSCREEN)
   {
       printf ("[start  read_parallel_port (), ");
       if (PARALLELPORTTRACE) printf ("PARALLELPORTTRACE is ON]\n");
       else                   printf ("PARALLELPORTTRACE is OFF]\n");
   }
   /* see initialize_adcs () for Init_PortA & B code                          */

#if defined (sgi)
#else
   /* Read PortA parallel port character by character for tactical orders   */
   /* reference:  Walt Landaker's mfi_a3.c in directory /h0/AUV and          */
   /*             page 3-12 of Motorola 6800 Series Manual for 6821 PIA      */
   /*             Programmable Interface Adapter.                             */

   /* Warning!  You may have to reset both computers to get the parallel    */
   /*           port to read & write properly.  Additionally,                */
   /*           on the 386 you can run PORTFIX to reset parallel port LPT1:  */

   temp = Read_PortA ((struct MFI_PIA *) MFI_BASE);      /* should clear busy! */

   index = 0;

   /* read port status (note sta not stb)                                     */
   PortAFlag = ck_sta ((struct MFI_PIA *) MFI_BASE);

   if (PARALLELPORTTRACE && DISPLAYSCREEN)
       printf ("\n    [time %5.2f read_parallel_port () resumed]", t);

   while (PortAFlag && 0x80)          /* see loop break for alternate exit    */
   {
       /* Note that ck_stb is used in mfi_a3 but ck_sta makes more sense      */
       PortAFlag = ck_sta ((struct MFI_PIA *) MFI_BASE); /* read port status */
       last_char = next_char;         /* read char and reset busy             */
       next_char = Read_PortA ((struct MFI_PIA *) MFI_BASE);

       if ((PortAFlag == 0x24) && (last_char == next_char)) break;

       /*  if next_char changed then flag may be messed up, read anyway       */
                                      /* check for ptr strobe                 */
                                      /* break => no character waiting        */
                                      /* control passes outside while loop    */

       else if (next_char == 13)      /* CR indicates end of line             */
       {
           current_command [index]   = 13;  /* CR    /n                        */
           current_command [index+1] = 10;  /* LF    extra, not needed         */
           current_command [index+2] =  0;  /* end of string delimiter         */
           index = 0;

           if (auvtextfile!=NULL)
           {
               fprintf (auvtextfile, "%s", current_command);
               fflush  (auvtextfile);
           }
           if (DISPLAYSCREEN)
           {
               printf ("\n\n>>> time %5.2f tactical message <<<\n", t);
               printf  ("%s", current_command);
               printf  ("\n");
           }
           /******** insert command processing logic here <<<<<<<<<<<<<<<<<< */
```

```
        }
        else if (next_char != 10)          /* LF ignored, others appended    */
        {
                current_command [index] = next_char;
                index ++;
                if (PARALLELPORTTRACE && DISPLAYSCREEN)
                {
                   /* print character to screen */
                      printf ("\n %c = %2x, PortAFlag after reading = %4x]  ",
                                  next_char, next_char, PortAFlag);
                }
        }
   } /* end while loop to read characters from port                          */

   if (PARALLELPORTTRACE && DISPLAYSCREEN)
   {
      printf("    [%c = %2x, PortAFlag = %2x, exiting read_parallel_port ()]",
             next_char, next_char, PortAFlag);
   }
#endif

   if (TRACE && DISPLAYSCREEN) printf ("[finish read_parallel_port ()]\n");

   return;

}  /* end read_parallel_port */

/****************************************************************************/

/* Card arrangement in AUV *PRIOR* to hardware upgrade 1993:

                 -------   -------   -----------------------
                 |     | |       | |          MFI          | <-- Direct
                 |     | |       | |      Reg A and B      |     Gyro
                 -------   -------   -----------------------

                       -----------------------
                       |        adc-2        | <-- Sonars
                       |      ( 0 - 15 )     |
                       -----------------------

  Main motors --> |  ada-1 dac   | |        ada-1 adc        | <-- Gyros
                  |  ( 0 - 3 )   | |       ( 0 - 15 )        |
                  --------------   -----------------------

                       -----------------------
                       |       dac-2b        | <-- Planes
                       |      ( 0 - 7 )      |
                       -----------------------


     This file contains the functions which address the A/D D/A cards
     directly in terms of voltages. */


  /**************************************************************************
   *  dac1(s,ch) -- writes signal 's' to ada-1 dac channel 'ch'
   *                    (allowable channels 0-3)
   **************************************************************************/
  void dac1(s,ch)
  int s,ch;
   {
if (NOT_YET_REIMPLEMENTED)
{

     ch = ch << 2;                          /* offset for G-96 addressing    */
```

```
    dac1_a[ch] = s >> 2;                      /* write upper 8 bits to MSB     */
    dac1_a[ch + DAC_LSB_OFFSET] = s << 6;     /* write lower 2 bits B3,B2      */
}
    return;
  } /* dac1 */

  /*************************************************************************
   *  dac2b(s,ch) -- writes signal 's' to dac2b dac channel 'ch'
   *                 (allowable channels 0-15)
   *************************************************************************/

  void dac2b(s,ch)
  int s,ch;
  {
if (NOT_YET_REIMPLEMENTED)
{

    ch = ch << 2;                              /* offset for G-96 addressing  */
    dac2b_a[ch] = s >> 2;                      /* write upper 8 bits to MSB   */
    dac2b_a[ch + DAC_LSB_OFFSET] = s << 6;     /* write lower 2 bits B3,B2    */
}
    return;
  } /* dac2b */

  /*************************************************************************
   *  adc1(n) -- reads ada-1 adc channel 'n' (channels 0-15)
   *************************************************************************/
  int adc1(n)
  int n;
  {
    int val;

if (NOT_YET_REIMPLEMENTED)
{
    adc1_a[ADC1_CMD_REG] = n;
    while (adc1_a[ADC1_STATUS_REG] > 20);    /* wait for data */
    val = adc1_a[ADC1_MSB] << 2;
    val += adc1_a[ADC1_LSB] >> 6;
    return (val);

} else return (0);
  } /* adc1 */

  /*************************************************************************
   *  adc2(n,g); -- Reads adc-2 channel 'n' (0-15)
   *                with gain 'g' (0 to F => 0 - 1024)
   *************************************************************************/

  int adc2(n,g)
  int n,g;
  {
    int val;

if (NOT_YET_REIMPLEMENTED)
{

    adc2_a[ADC2_CH_GAIN] = (n << 4) | g;      /* set c&g, start conv */
    while((adc2_a[ADC2_STATUS_REG] & 0x7) != 0);  /* wait for ready */

  /* This adc uses 0 - 4095 to represent full scale input, in order
    to write to the dac (which uses 0 -1023 for full scale) you
    must divide val by 4 or shift right by 2. Use the next line to
    get full resolution.
    val = adc2_a[ADC2_DATA];
    The next line is used for testing purposes only

    val = adc2_a[ADC2_DATA] >> 2;  */
```

```
        val = adc2_a[ADC2_DATA];
        val = val & 0x0FFF;

        return(val);
    } else return (0);
  } /* adc2 */

/****************************************************************************/
/*
The program code for the Multi-Function Interface originated from 'mfi.c'
Routines include Init_PortA, Init_PortB, Read_PortA, Read_PortB, Read_PortAB

Excerpt of 'mfi.c' comments follows:

  Program example for the Multi-Function-Interface (MFI)
  This example uses the 6821 PIA on the MFI board
  General purpose functions are provided to initialize the PIA
  and read/write data to the ports

  MFI P2 connector definitions are provided by the GESMFI-1 data
  sheet available from GESPAC, Inc.

  6821 device specifics are covered in the 8-bit microprocessor
  & peripheral data book from Motorola Inc.

  3/1/91        J. Rawlins      RealTime Software Consulting

*/
/****************************************************************************/

  /**************************************************************************
   *  Init_PortA(base, dir) -- Initialize Port A of MFI
   *                  dir: 1 = output port, 0 = input port
   **************************************************************************/

void Init_PortA (base, dir)
  register struct MFI_PIA *base;/* base address of MFI board on G96 bus      */
  int dir;                      /* direction: 1 = output port, 0 = input port */
  {
    register short temp;
    temp = base->cra;    /* save contents of control reg. (no-op)           */
    base->cra = 0x00;    /* select:  b2 = 0  data direction reg.            */
    base->pra = 0x00;    /* set portA: 0 = input                            */
    base->cra = 0x24;    /* select: access data reg.s (b2=1)                */
                         /*  b5=1,b4=0,b3=0(read  w/ca1 restore)            */
}/* Init_PortA */


  /**************************************************************************
   *  Init_PortB(base, dir) -- Initialize Port B of MFI
   *                  dir: 1 = output port, 0 = input port
   **************************************************************************/

void Init_PortB (base, dir)

register struct MFI_PIA *base;  /* base address of MFI board on G96 bus      */
int dir;                        /* direction: 1 = output base, 0 = input base */
{
        register short temp;

        temp = (base->crb & 0x00FF);  /* get current value of control A     */
        temp &= ~4;                   /* clear bit #2 so we can access ddra  */
        base->crb = temp;
        if ( dir )                    /* make port B all outputs             */
                base->prb = 0x00FF;
        else                          /* port B is all inputs                */
```

```
            base->prb = 0x0000;
        temp |= 4;                        /* set bit #2 to access data registers  */
        base->crb = temp;
}/* Init_PortB */

/**************************************************************************
 *  Read_PortA  (base) -- returns 8 bit value from port A
 *
 *************************************************************************/

unsigned char Read_PortA (base)

register struct MFI_PIA *base;              /* base address of MFI              */
{
        register unsigned short temp;
        temp = base->pra;                   /* read data reg.should reset busy */
        return(temp & 0x00FF);              /* return data to calling program  */
}

/**************************************************************************
 *  Read_PortB  (base) -- returns 8 bit value from port B
 *
 *************************************************************************/

unsigned char Read_PortB (base)
register struct MFI_PIA *base;                          /* base address of MFI */
{
        register unsigned short temp;
        temp = base->prb;
        return(temp & 0x00FF);
}

/**************************************************************************
 *  Read_PortAB (base) -- return a 16 bit value from ports
 *                        A and B combined then mask off
 *                        the 15 th and 16 th bits.
 * Note: PIA PA0-PA7 is the LSB and PB0-PB7 the MSB
 *************************************************************************/

unsigned short  Read_PortAB (base)

register struct MFI_PIA *base;      /* base address of MFI                    */
{
        register unsigned short hi,lo,temp;

        lo = (base->pra & 0x00FF);  /* get least significant byte from A      */
        hi = (base->prb & 0x00FF);  /* and most  significant byte from B      */
        temp = ((hi << 8) + lo);    /* shift hi into upper byte of word       */
        return ( temp );            /* return data                            */
}

/***************************************************************************/

void set_bsyA(base)  /* sets CB2 high (for busy to sending port)              */

register struct MFI_PIA *base;  /* base address of MFI                        */
{
    register short temp;
    temp = (base->cra & 0xFF);  /* save cra values                            */
    base->cra = 0x38;           /* 8 bit 1=  CR2  high                        */
    base->cra = temp;           /* restore cra values                         */
}
/* sets CB2 low (for -busy to sending port) */
void rst_bsyA(base)
register struct MFI_PIA *base;  /* base address of MFI                        */
{
    register short temp;
```

```
   temp = (base->cra & 0xFF);  /* save cra values                         */
   base->cra = 0x30;           /* 8 bit 0= CR2 low                        */
   base->cra = temp;           /* restore cra values                      */

int ck_sta (base)
register struct MFI_PIA *base;  /* base address of MFI                    */
{
   register unsigned short temp;
   temp = base->cra;           /* save cra values                         */
   return (temp);
}

/*****************************************************************************/

void center_sonar ()
{
   int  direction,encoder_width;
   char encode;

   if (! SONARINSTALLED)
   {
      printf ("[start/stop center_sonar, SONARINSTALLED false]\n");
      return;
   }

   if (TRACE && DISPLAYSCREEN) printf ("[start  center_sonar ()]\n");

   encoder_width = 0;
   direction = 1;

   /* set_step_size('H'); */          /* '1' = 0.9, '2' = 1.8, '4' = 3.6 */

   /* Are we inside the Encoder Sensor ? */
   encode = query_sonar_1_reply ('M');    /* Test Head Direction (No Step) */
   if (SONARTRACE && DISPLAYSCREEN)
      printf("center_sonar:  encode = %c\n",encode);

   if ((encode == 't') || (encode == 'T'))
   {
      while( (encode == 't') || (encode == 'T') )
      {
         encode = query_sonar_1_reply ('+'); /* Index Sonar '+' direction */
      }

      /* Outside Encoder Sensor Now */
      direction = -1; /* Reverse Sonar Rotation to Establish Encoder Width */
   }

   while( (encode == 'f') || (encode == 'F') )
   {
      if(direction == 1)
      {
         encode = query_sonar_1_reply ('+'); /* Index Sonar '+' direction */
         if (SONARTRACE && DISPLAYSCREEN) printf("%c\n",encode);
      }
      else
      {
         encode = query_sonar_1_reply ('-'); /* Index Sonar '-' direction */
      }
   }

   /* Found Edge of Encoder */
   while( (encode == 't') || (encode == 'T') )
   {
      encoder_width = encoder_width + 1;

      if(direction == 1)
```

```
      {
         encode = query_sonar_1_reply ('+'); /* Index Sonar '+' direction */
      }
      else
      {
         encode = query_sonar_1_reply ('-'); /* Index Sonar '-' direction */
      }
   }
   if (SONARTRACE && DISPLAYSCREEN)
      printf ("center_sonar:  encoder width = %d\n",encoder_width);

   if (TRACE && DISPLAYSCREEN) printf ("[finish center_sonar ()]\n");

   return;

} /* end center_sonar () */

/*****************************************************************************/

char query_sonar_1_reply (command_char)

   char command_char;

{
   /* code tested & taken from headtest.c (prior version ahead.c)            */

   int index,n,n_bytes;
   char reply,xx[20],c[1];

   if (! SONARINSTALLED)
   {
      printf ("[start/stop query_sonar_1_reply (), SONARINSTALLED false]\n");
      reply = ' ';
      return (reply);
   }

   if (TRACE && DISPLAYSCREEN) printf ("[start  query_sonar_1_reply ()]\n");

   /* likely string problems here <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< */
   c[0] = command_char;
   if (SONARTRACE && DISPLAYSCREEN)
      printf ("query_sonar_1_reply:  command_char = %c\n", command_char);
   n    = write (sonarpath,c,1);  /* write characters to sonarpath /t3 device */
   tsleep(10);

   n_bytes = _gs_rdy (sonarpath);
   if (SONARTRACE && (n_bytes > 1) && DISPLAYSCREEN)
      printf ("query_sonar_1_reply:  lost reply data, n_bytes = %d\n",n_bytes);

   if (n_bytes <= 0)
   {
      printf("query_sonar_1_reply:  bad read, n_bytes = %d\n",n_bytes);
   }
   else
   {
      n = read (sonarpath,xx,n_bytes);  /* n unused?????? */

      if (SONARTRACE && (n_bytes > 0) && DISPLAYSCREEN)
      {
         for (index = 0; index < n_bytes; index++)
         {
              printf("[%c %2d %2x]  ", xx[index], xx[index], xx[index]);
              if ((index+1) % 5 == 0)
                   printf("\n"); /* prevent writing off screen */
         }
         printf("\n");
```

```
        }
    }
    reply = xx[0];

    if (TRACE && DISPLAYSCREEN)
        printf ("[finish query_sonar_1_reply () returns %c]\n", reply);

    return (reply);
}

/****************************************************************************/

void set_step_size (step_code)

    char step_code;
{
    unsigned int n;
    char        reply;

    if (! SONARINSTALLED)
    {
        printf ("[start/stop set_step_size (), SONARINSTALLED false]\n");
        return;
    }

    if (TRACE && DISPLAYSCREEN) printf ("[start  set_step_size ()]\n");

    if (SONARTRACE && DISPLAYSCREEN) printf ("step code = %c\n",step_code);
    write (sonarpath,step_code,1);
    tsleep (1);  /* 256ths of a second */

    n = read (sonarpath, reply, 1);    /* n unused?????? */

    if (SONARTRACE && DISPLAYSCREEN) printf("step = %c\n",reply);

    if (TRACE && DISPLAYSCREEN) printf ("[finish set_step_size ()]\n");

    return;
}

/****************************************************************************/

void tty_mode (tty_mode_path, mode)

    int   tty_mode_path;
    int   mode;              /* note type specifications differ from headtest.c */
{
    static struct sgbuf old,new;
    static int init = 1;
    int status;

    if (! SONARINSTALLED)
    {
        if (TRACE && DISPLAYSCREEN)
            printf ("[start/stop  tty_mode, SONARINSTALLED false ()]\n");
        return;
    }

    if (init)
    {
       if (TRACE && DISPLAYSCREEN) printf ("[start  tty_mode ()]\n");

       init = 0;
       status = _gs_opt(tty_mode_path, &old);
       status = _gs_opt(tty_mode_path, &new);

       new.sg_class  = 0;
```

```
        new.sg_case    = 0;
        new.sg_backsp  = 0;
        new.sg_delete  = 0;
        new.sg_echo    = 0;
        new.sg_alf     = 0;
        new.sg_nulls   = 0;
        new.sg_pause   = 0;
        new.sg_page    = 0;
        new.sg_bspch   = 0;
        new.sg_dlnch   = 0;
        new.sg_eorch   = 0;
        new.sg_eofch   = 0;
        new.sg_rlnch   = 0;
        new.sg_dulnch  = 0;
        new.sg_psch    = 0;
        new.sg_kbich   = 0;
        new.sg_kbach   = 0;
        new.sg_bsech   = 0;
        new.sg_bellch  = 0;
        new.sg_parity  = 0;
        new.sg_tabcr   = 0;
        new.sg_tabsiz  = 0;
        new.sg_tbl     = 0;
        new.sg_col     = 0;
        new.sg_err     = 0;
    }

    if   (mode) _ss_opt (tty_mode_path, &new);
    else        _ss_opt (tty_mode_path, &old);

    if (TRACE && DISPLAYSCREEN) printf ("[finish tty_mode ()]\n");

    return;
}

/*************************************************************************************/

void print_valid_keywords ()
{
printf ("\n");
printf ("                    [help] [trace|notrace] [loopforever|looponce]\n");
printf ("                    [wait #] [time #] [timestep (0.0..5.0)] [mission]\n");
printf ("                    [keyboard|keyboard-off] [quit] [kill]\n");
printf ("                    [rpm] [course] [depth] [thrusters|thrusters-off]\n");
printf ("                    [loopfilebackup] [entercontrolconstants]\n");
printf ("                    [rotate] [position] [orientation]\n");
printf ("                    [sonartrace|sonartraceoff] [sonarinstalled]\n");
printf ("                    [trace|trace-off] [parallelporttrace] \n");
printf ("                    [remotehost hostname][realtime|nopause] [pause]\n");
printf ("                    [loop-forever|loop-once][entercontrolconstants]\n\n");
printf ("                    [silence][e-mail|no-email] [waypoint]\n\n");
printf ("See ~/execution/mission.script.HELP for command syntax details.\n");
printf ("\n");

#if defined(sgi)
   printf ("popping up 'mission.script.HELP' as a zip file...\n");
   system ("zip -v ~/execution/mission.script.HELP");
#endif

   return;

} /* end print_valid_keywords */

/*************************************************************************************/

void open_virtual_world_socket ()        /* see os9sender.c for original code */
```

```
{
  if (TRACE && DISPLAYSCREEN)
      printf ("[start open_virtual_world_socket ()]\n");

/* Initialize communications blocks                                          */

/* Initialize both client & server ********************************************/

   /* Signal handlers for termination to override net_open () and net_close ()*/
   /*    signal handlers.  Otherwise you are unable to ^C kill this program.  */

#if defined(sgi)
signal (SIGHUP,  shutdown_virtual_world_socket);/* hangup                     */
signal (SIGINT,  shutdown_virtual_world_socket);/* interrupt character        */
signal (SIGKILL, shutdown_virtual_world_socket);/* kill signal from Unix      */
signal (SIGPIPE, shutdown_virtual_world_socket);/* broken pipe from other host*/
signal (SIGTERM, shutdown_virtual_world_socket);/* software termination       */
#endif

/* Initialize sender *********************************************************/

   /* start by finding default/desired remote host to connect to             */
   {

      server_entity = gethostbyname (virtual_world_remote_host_name);
      if (server_entity == NULL)
      {
        if (DISPLAYSCREEN)
        {
          printf("[open_virtual_world_socket:  virtual world remote host\n");
          printf("  (\"%s\") not found]\n",
                  virtual_world_remote_host_name);
          fflush (stdout);
        }
        /* error message needed on (open) output file   <<<<<<<<<<<<<<<<<<<< */
        virtual_world_socket_opened = FALSE;
        return;
      }
      else if (TRACE && DISPLAYSCREEN)
           {
              printf("[open_virtual_world_socket:  virtual world remote host ");
              printf("(\"%s\") located]\n",  virtual_world_remote_host_name);
           }

      /* Client opens server port ********************************************/

      /* Fill in structure 'server_address' with the address of the          */
      /*         remote host (i.e. SERVER) that we want to connect with:     */

#if defined(sgi)
      bzero ((char *) &server_address, sizeof (server_address));
#endif
      server_address.sin_family    = AF_INET;      /* Internet protocol family */

      /* copy server IP address into sockaddr_in struct server_address       */
#if defined(sgi)
      bcopy (server_entity->h_addr, &(server_address.sin_addr.s_addr),
             server_entity->h_length);
#else
      strncpy(&(server_address.sin_addr.s_addr), server_entity->h_addr,
             server_entity->h_length);
#endif

      /* make sure port is in network byte order                             */
      server_address.sin_port = htons (AUVSIM1_TCP_PORT_1);

      /* Open TCP (Internet stream) socket                                   */
```

```
      if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
      {
         if (DISPLAYSCREEN)
         {
             printf ("[open_virtual_world_socket:  client can't open server");
             printf (" virtual world stream socket]");
         }
         /* error message needed on (open) output file <<<<<<<<<<<<<<<<<<<<<< */
         virtual_world_socket_opened = FALSE;
         return;
      }
      else if (TRACE && DISPLAYSCREEN)
           {
              printf ("[open_virtual_world_socket:  client opened");
              printf (" virtual world server socket successfully]\n");
           }

      /* Connect to the server.  Process will block/sleep until connection is
           is established.  Timeout will return an error.                     */
      if (connect (                   socket_descriptor,
                  (struct sockaddr *) &server_address,
                                 sizeof (server_address)) < 0)
      {
         if (DISPLAYSCREEN)
         {
             printf ("[open_virtual_world_socket:  client can't connect to");
             printf (" virtual world server socket]\n");
         }
         /* error message needed on (open) output file <<<<<<<<<<<<<<<<<<<<<< */
         virtual_world_socket_opened = FALSE;
         return;
      }
      else if (TRACE && DISPLAYSCREEN)
           {
              printf ("[execution client connected to virtual world");
              printf (" server socket successfully]\n");
           }
      virtual_world_socket_opened = TRUE;

   } /* end initialization                                                     */

   socket_stream = socket_descriptor;    /* client */

   if (TRACE && DISPLAYSCREEN)  /* print final info */
   {
     printf("[open_virtual_world_socket CLIENT:  socket_descriptor = %d]\n",
                                                 socket_descriptor);
     printf("[                                   socket_accepted   = %d]\n",
                                                 socket_accepted);
     printf("[                                   socket_stream     = %d]\n",
                                                 socket_stream);
   }

   if (TRACE && DISPLAYSCREEN)
       printf ("[finish open_virtual_world_socket ()]\n");

   return;

}/* end open_virtual_world_socket () */

/*****************************************************************************/

void shutdown_virtual_world_socket ()     /* see os9sender.c for original code */
{
   int  kill_return_value;

   shutdown_signal_received = TRUE;
```

```
    if (virtual_world_socket_opened == FALSE)
    {
        if (TRACE && DISPLAYSCREEN)
        {
           printf ("[virtual_world_socket_opened FALSE,");
           printf (" shutdown_virtual_world_socket ignored]\n");
        }
        return;
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[shutdown_virtual_world_socket start ...]\n");

    /* No need to send a message to other side that bridge is going down,      */
    /*    since SIGPIPE signal trigger may shutdown server on other side        */

    if (close (socket_stream) == -1)
    {
        if (TRACE && DISPLAYSCREEN)
         printf ("shutdown_virtual_world_socket close (socket_stream) failed\n");

        /* shutdown () reference:  "Using OS-9 Internet" manual p. 2-55         */

        if (shutdown (socket_stream, 2) == -1)
        {
            if (TRACE && DISPLAYSCREEN)
            {
                printf ("[shutdown_virtual_world_socket shutdown");
                printf (" (socket_stream, 2) failed]\n");
            }

            kill_return_value = kill (socket_stream, SIGKILL);

            if (TRACE && DISPLAYSCREEN)
            {
                printf ("[shutdown_virtual_world_socket kill (socket_stream,");
                printf (" SIGKILL) returned %d]\n", kill_return_value);
            }
        }
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[shutdown_virtual_world_socket return]\n");

    return;

} /* end shutdown_virtual_world_socket () */

/*****************************************************************************/

void send_data_on_virtual_world_socket ()/* see os9sender.c for original code */
{
   bytes_left          = socket_length;
   bytes_written       = 0;
   ptr_index           = buffer;  /* this global string is the data to be sent  */

   if (virtual_world_socket_opened == FALSE)
   {
       return;
   }
   if (TRACE && DISPLAYSCREEN)
       printf ("[send_data_on_virtual_world_socket start ...]\n");

   while ((bytes_left > 0) && (bytes_written >= 0))  /* write loop ***********/
   {
       bytes_sent = write (socket_stream, ptr_index, bytes_left);

       if       (bytes_sent <  0) bytes_written = bytes_sent;
       else if (bytes_sent >  0)
```

```
            {
                bytes_left     -= bytes_sent;
                bytes_written  += bytes_sent;
                ptr_index      += bytes_sent;
            }
        if (LOCATIONLAB && TRACE && DISPLAYSCREEN)
        {
              printf ("[record_data send_telemetry_to_server loop");
              printf ("  bytes sent = %d]\n", bytes_sent);
        }
    }
    if    (bytes_written < 0)
    {
        if (LOCATIONLAB && DISPLAYSCREEN)
        {
          printf ("[record_data send_telemetry_to_server () send failed, ");
          printf ("%d bytes_written]\n", bytes_written);
        }
        /* error message needed on (open) output file   <<<<<<<<<<<<<<<<<<<<<< */
    }
    else if (LOCATIONLAB && TRACE && DISPLAYSCREEN)
         {
             printf ("[record_data send_telemetry_to_server total bytes sent");
             printf (" = %d]\n", bytes_written);
         }

    /* Check termination ******************************************************/

    if (strncmp (buffer, "shutdown", 8) == 0)
    {
        if (TRACE && DISPLAYSCREEN) printf
             ("[send_data_on_virtual_world_socket: shutdown_signal_received]\n");
        shutdown_virtual_world_socket ();
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[send_data_on_virtual_world_socket return]\n");

    return;

} /* end send_data_on_virtual_world_socket () */

/**************************************************************************/

void get_stream_from_virtual_world_socket ()   /* see os9sender.c for original */
{
   if (virtual_world_socket_opened == FALSE)
   {
       return;
   }
   if (TRACE && DISPLAYSCREEN)
       printf ("[get_stream_from_virtual_world_socket start ...]\n");

   /* listen to remote host, relay to local network/program                 */

   bytes_left        = socket_length;
   bytes_received    = 0;
   ptr_index         = buffer_received;

   while ((bytes_left > 0) && (bytes_received >= 0))   /* read loop ************/
   {
      bytes_read = read (socket_stream, ptr_index, bytes_left);

      if      (bytes_read <  0) bytes_received = bytes_read;
      else if (bytes_read >  0)
      {
           bytes_left     -= bytes_read;
           bytes_received += bytes_read;
```

```
            ptr_index       += bytes_read;
        }
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[get_stream_from_virtual_world_socket receiver block");
            printf (" loop bytes_read = %d]\n", bytes_read);
        }
        /* if nothing is waiting to be read, break out of read loop                */
        if ((bytes_read == 0) && (bytes_received == 0)) break;
    }
    if  (bytes_received < 0)        /* failure      */
    {
        if (TRACE && DISPLAYSCREEN)
        {
            printf ("[get_stream_from_virtual_world_socket receiver block read");
            printf (" failed, bytes_received = %d\n", bytes_received);
        }
    }
    else if (bytes_received == 0)   /* no transfer */
    {
        if (TRACE && DISPLAYSCREEN)
        {
          printf("[get_stream_from_virtual_world_socket received 0 bytes!!]\n");
        }
    }
    else if (bytes_received > 0)   /* success      */
        {
            if (TRACE && DISPLAYSCREEN)
            {
            printf("[get_stream_from_virtual_world_socket received %d bytes]\n",
                      bytes_received);
            }
        }

    /* Check termination ********************************************************/

    if (strncmp (buffer_received, "shutdown", 8) == 0)
    {
        if (TRACE && DISPLAYSCREEN) printf
            ("[get_data_on_virtual_world_socket: shutdown_signal_received]\n");
        shutdown_virtual_world_socket ();
    }
    if (TRACE && DISPLAYSCREEN)
        printf ("[get_stream_from_virtual_world_socket return]\n");

    return;

} /* end get_stream_from_virtual_world_socket () */

/****************************************************************************/

double degrees (rads)      /* radians input */
       double  rads;
{
   return rads * 180.0 / PI;
}
/****************************************************************************/

double radians (degs)      /* degrees input*/
       double  degs;
{
   return degs * PI / 180.0;
}
/****************************************************************************/

double normalize (degs)      /* degrees input*/
```

```
        double  degs;
{
   double result = degs;

   while (result <    0.0) result += 360.0;
   while (result >= 360.0) result -= 360.0;

   return result;
}
/*****************************************************************************/

double normalize2 (degs)      /* degrees input*/
        double  degs;
{
   double result = degs;

   while (result <= -180.0) result += 360.0;
   while (result >   180.0) result -= 360.0;

   return result;
}
/*****************************************************************************/

double radian_normalize (rads)      /* radians input*/
        double  rads;
{
   double result = rads;

   while (result <       0.0) result += 2.0 * PI;
   while (result >= 2.0 * PI) result -= 2.0 * PI;

   return result;
}
/*****************************************************************************/

double radian_normalize2 (rads)      /* radians input*/
        double  rads;
{
   double result = rads;

   while (result <= - PI) result += 2.0 * PI;
   while (result >    PI) result -= 2.0 * PI;

   return result;
}
/*****************************************************************************/

void clamp (clampee, absolute_min, absolute_max, name)
            double * clampee;
            double   absolute_min;
            double   absolute_max;
            char   * name;
{
   double new_value, local_min, local_max;

   if ((absolute_max == 0.0) && (absolute_min == 0.0)) return;    /* no clamp */

   if (absolute_max >= absolute_min) /* ensure min & max used in proper order */
   {
       local_min = absolute_min;
       local_max = absolute_max;
   }
   else
   {
       local_min = absolute_max;
       local_max = absolute_min;
   }
```

```
    if ((* clampee) > local_max)
    {
        new_value = local_max;

        if (TRACE && DISPLAYSCREEN)
            printf ("[clamping %s from %5.3f to %5.3f]\n",
                               name, * clampee, new_value);

        * clampee = new_value;
    }
    if ((* clampee) < local_min)
    {
        new_value = local_min;

        if (TRACE && DISPLAYSCREEN)
            printf ("[clamping %s from %5.3f to %5.3f]\n",
                               name, * clampee, new_value);

        * clampee = new_value;
    }
}
/******************************************************************************/
#if defined(sgi)
#else

/* thanks to Michael Olberg    Oct 20, 94    olberg@bele.oso.chalmers.se      */

double atan2(y, x)
             double y; double x;
{
    if (TRACE && DISPLAYSCREEN)
            printf ("[atan2 (%5.3f, %5.3f)]\n", y, x);

    if (x == 0.0) {
        if (y < 0.0)       return(-PI/2.0);
        else               return( PI/2.0);
    } else {
        if (x < 0.0) {
            if (y < 0.0)   return(atan(y/x)-PI);
            else           return(atan(y/x)+PI);
        } else             return(atan(y/x));
    }
}
    /* as to the tanh you will simply have to use */

double sinh(x)
       double x;
{
    return (exp(x) - exp(-x))/2.0;
}

double cosh(x)
       double x;
{
    return (exp(x) + exp(-x))/2.0;
}

double tanh(x)
       double x;
{
    return sinh(x)/cosh(x);
}

#endif

/******************************************************************************/
```

```
double sign (x)
       double x;
{
    if       (x > 0.0) return  1.0;
    else if  (x < 0.0) return -1.0;
    else               return  0.0;
}

/******************************************************************************/
/******************************************************************************/
/* end of execution.c                                                         */
/******************************************************************************/
/******************************************************************************/
```

D. *parse_functions.c* Tactical Script Command Parse Functions

```
/******************************************************************************/
/*
 Program:          parse_functions.c

 Authors:          Don Brutzman

 Revised:          28 October 94

 System:           AUV Gespac 68020/68030, OS-9 version 2.4
 Compiler:         Gespac cc Kernighan & Richie (K&R) C

 Compilation:      ftp>      put parse_functions.c
                   auvsim1> chd execution
     [68020]       auvsim1> make -k2f execution
     [68030]       auvsim1> make      execution

     [Irix ]       fletch>  make execution

 Purpose:          Reduce size of execution.c to allow OS-9 C compiler to work


*/
/******************************************************************************/
/* parse_functions.c */

#include "globals.h"
#include "defines.h"

   void            parse_command_line_flags              ();
   void            parse_mission_script_commands         ();
   void            parse_mission_string_commands         ();

extern void        print_valid_keywords                  ();
extern void        send_data_on_virtual_world_socket     ();
extern void        get_control_constants                 ();
extern void        clamp                                 ();

/******************************************************************************/

void parse_command_line_flags (argc, argv)

  int argc; char **argv;     /* command line arguments */
{
   int index;

   if (DISPLAYSCREEN)
   {
       printf ("[parse_command_line_flags start: # arguments = %d]\n[", argc);
       for (i = 0; i < argc; i++) printf (" %s", argv[i]);
       printf (" ]\n");
   }

   if (DISPLAYSCREEN) printf ("[parse arguments:  ");
   {
      for (i = 1; i < argc; i++)
      {
        printf ("%s ", argv[i]);

        for (index = 0; index <= (int)strlen (argv[i]); index++)/* uppercase */
             argv[i] [index] = toupper (argv[i] [index]);
      }
        printf ("]\n");
```

```
}

strcpy (buffer, ""); /* initialize for SILENT */

for (i = 1; i < argc; i++)
{
   if      ((strcmp (argv[i],  "HELP") == 0) ||
             (strcmp (argv[i],     "?") == 0) ||
             (strcmp (argv[i],    "/?") == 0) ||
             (strcmp (argv[i],    "-?") == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[print_help] ");
         print_help = TRUE;
   }
   else if ((strcmp (argv[i],  "KEYBOARD")       == 0) ||
             (strcmp (argv[i], "KEY-BOARD")      == 0) ||
             (strcmp (argv[i], "KEYBOARD-INPUT") == 0) ||
             (strcmp (argv[i], "KEYBOARDINPUT")  == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[KEYBOARDINPUT = TRUE] ");
         KEYBOARDINPUT = TRUE;
   }
   else if (strcmp (argv[i],  "TRACE") == 0)
   {
         if (TRACE && DISPLAYSCREEN) printf ("[TRACE = TRUE] ");
         TRACE = TRUE;
   }
   else if ((strcmp (argv[i],  "TRACEOFF") == 0) ||
             (strcmp (argv[i], "TRACE-OFF") == 0) ||
             (strcmp (argv[i],   "NOTRACE") == 0) ||
             (strcmp (argv[i],  "NO-TRACE") == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[TRACE = FALSE] ");
         TRACE = FALSE;
   }
   else if ((strcmp (argv[i],  "LOOPFOREVER") == 0) ||
             (strcmp (argv[i], "LOOP-FOREVER") == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[LOOPFOREVER] ");
         LOOPFOREVER = TRUE;
   }
   else if ((strcmp (argv[i],  "LOOPONCE") == 0) ||
             (strcmp (argv[i], "LOOP-ONCE") == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[LOOPONCE] ");
         LOOPFOREVER = FALSE;
   }
   else if ((strcmp (argv[i],   "LOOPFILEBACKUP") == 0) ||
             (strcmp (argv[i], "LOOP-FILE-BACKUP") == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[LOOPFILEBACKUP] ");
         LOOPFILEBACKUP = TRUE;
   }
   else if ((strcmp (argv[i],   "ENTERCONTROLCONSTANTS") == 0) ||
             (strcmp (argv[i], "ENTER-CONTROL-CONSTANTS") == 0))
   {
         if (TRACE && DISPLAYSCREEN) printf ("[ENTERCONTROLCONSTANTS] ");
         ENTERCONTROLCONSTANTS = TRUE;
   }
   else   if ((strcmp (argv[i],  "TACTICAL")    == 0) ||
               (strcmp (argv[i],  "TACTICAL-ON") == 0))
   {
         printf ("[%s]\n", argv[i]);
         TACTICAL = TRUE;
   }
   else   if ((strcmp (argv[i],  "NO-TACTICAL")  == 0) ||
               (strcmp (argv[i],  "TACTICAL-OFF") == 0))
```

```
{
        printf ("[%s]\n", argv[i]);
        TACTICAL = FALSE;
}
else if ((strcmp (argv[i],  "SONARTRACE") == 0) ||
         (strcmp (argv[i], "SONAR-TRACE") == 0))
{
        if (TRACE && DISPLAYSCREEN) printf ("[SONARTRACE] ");
        SONARTRACE = TRUE;
}
else if ((strcmp (argv[i],   "SONARTRACEOFF") == 0) ||
         (strcmp (argv[i], "SONAR-TRACE-OFF") == 0))
{
        if (TRACE && DISPLAYSCREEN) printf ("[SONARTRACEOFF] ");
        SONARTRACE = FALSE;
}
else if ((strcmp (argv[i],  "SONARINSTALLED") == 0) ||
         (strcmp (argv[i], "SONAR-INSTALLED") == 0))
{
        if (TRACE && DISPLAYSCREEN) printf ("[SONARINSTALLED] ");
        SONARINSTALLED = TRUE;
}
else if ((strcmp (argv[i],    "PARALLELPORTTRACE") == 0) ||
         (strcmp (argv[i], "PARALLEL-PORT-TRACE") == 0))
{
        if (TRACE && DISPLAYSCREEN) printf ("[PARALLELPORTTRACE] ");
        PARALLELPORTTRACE = TRUE;
}
else if ((strcmp (argv[i],  "SILENT") == 0) ||
         (strcmp (argv[i], "SILENCE") == 0))
{
        if (TRACE && DISPLAYSCREEN) printf ("[SILENT] ");
        /* send to virtual world after socket is open */
        strcpy (buffer, "SILENT"); /* copy current command to buffer */
}
else if ((strcmp (argv[i],  "TIMESTEP") == 0) ||
         (strcmp (argv[i], "TIME-STEP") == 0))
{
        i++;
        if (i >= argc) print_help = TRUE;
        else
        {
            sscanf (argv[i], "%F", &TIMESTEP);
            if (TRACE && DISPLAYSCREEN) printf("[TIMESTEP %f]", TIMESTEP);
            if (TIMESTEP > 0.0) dt = TIMESTEP;
            else if (TRACE && DISPLAYSCREEN)
                    printf("  illegal TIMESTEP value, ignored.");
            if (TRUE && DISPLAYSCREEN) printf("  [dt =  %f]", dt);
        }
}
else if ((strcmp (argv[i],  "REMOTEHOST") == 0)  ||
         (strcmp (argv[i], "REMOTE-HOST") == 0)  ||
         (strcmp (argv[i],  "REMOTE")     == 0)  ||
         (strcmp (argv[i],  "HOST")       == 0))
{
        i++;
        if (i >= argc) print_help = TRUE;
        else
        {
            sscanf (argv[i], "%s",  virtual_world_remote_host_name);
            if (TRACE && DISPLAYSCREEN)
                printf("[REMOTE-HOST %s]", virtual_world_remote_host_name);
        }
}
else if ((strcmp (argv[i],  "REALTIME") == 0) ||
         (strcmp (argv[i], "REAL-TIME") == 0))
{
```

```
        if (TRACE && DISPLAYSCREEN) printf ("[REALTIME] ");
        REALTIME = TRUE;
    }
    else if ((strcmp (argv[i],  "NOREALTIME")   == 0)  ||
             (strcmp (argv[i],  "NO-REALTIME")  == 0)  ||
             (strcmp (argv[i],  "NO-REAL-TIME") == 0)  ||
             (strcmp (argv[i],  "NOWAIT")       == 0)  ||
             (strcmp (argv[i], "NO-WAIT")       == 0)  ||
             (strcmp (argv[i],  "NOPAUSE")      == 0)  ||
             (strcmp (argv[i], "NO-PAUSE")      == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[NOWAIT] ");
        REALTIME = FALSE;
    }
    else if ((strcmp (argv[i],  "NOEMAIL")  == 0) ||
             (strcmp (argv[i],  "NO-EMAIL") == 0))
    {
        if (TRACE && DISPLAYSCREEN) printf ("[NO EMAIL] ");
        EMAIL = FALSE;
    }
    else  print_help = TRUE;  /* invalid command line entry parameter found */

  } /* end for loop through command line parameters                          */

  if (print_help) /* print help string ***************************************/
  {
   printf("\nUsage:  execution \n");
   print_valid_keywords ();
   exit (-1);
  }

  if (TRACE && DISPLAYSCREEN) printf ("\n[parse_command_line_flags complete]\n");

  return;

} /* end parse_command_line_flags () */

/*****************************************************************************/

void parse_mission_script_commands ()  /* get data from file at program start */
                                       /* mission.script.HELP => descriptions */
{
    int    index, read_another_line, parameters_read;
  double   parameter1, parameter2, parameter3, parameter4, parameter5;
    char   backupcommand [50], new_filename [30];

   if (TRACE && DISPLAYSCREEN)
       printf ("[start parse_mission_script_commands ()]\n");

   if   ((auvscriptfile == NULL) || feof (auvscriptfile) || auvscriptfilequit)
   {
        if (DISPLAYSCREEN)
        {
           printf ("[opening a copy of the auvscriptfile %s]\n",
                   AUVSCRIPTFILENAME);
           fflush (stdout);
        }
#if defined(sgi)
        sprintf (backupcommand, "rm      mission.script.backup");
         printf ("%s\n", backupcommand);
        system  (backupcommand);
        sprintf (backupcommand, "cp   %s mission.script.backup",
                 AUVSCRIPTFILENAME);
         printf ("%s\n", backupcommand);
        system  (backupcommand);
#else
        /* OS-9 */
```

```
        sprintf (backupcommand, "del     mission.script.backup");
         printf ("%s\n", backupcommand);
        system  (backupcommand);
        sprintf (backupcommand, "copy %s mission.script.backup",
                 AUVSCRIPTFILENAME);
         printf ("%s\n", backupcommand);
        system  (backupcommand);
#endif

        auvscriptfile = fopen ("mission.script.backup","r");  /* input file */
        if (auvscriptfile == NULL)
        {
           printf ("AUV execution:  script file %s\n", AUVSCRIPTFILENAME);
           printf
       ("                (or backup copy mission.script.backup) not found.\n");
           printf ("                Ensure you are in the right directory:\n");
           printf ("                auvsim1> chd /h0/execution      or\n");
           printf ("                unix>    cd  ~brutzman/execution\n");
           printf ("Exit.\n");
           exit (-1);
        }
        auvscriptfilequit = FALSE;

        sprintf (buffer, "%s.backup", AUVORDERSFILENAME);
        auvordersfile = fopen (buffer,"w");  /* output file */
        if (TRACE && DISPLAYSCREEN)
            printf ("auvordersfile (%s) = %x\n", AUVORDERSFILENAME,
                     auvordersfile);
        if (auvordersfile == NULL)
        {
           printf ("AUV execution:  %s file not opened.\n", buffer);
           printf ("                Error.\n");
           printf ("Exit.\n");
           exit (-1);
        }
        if (TRACE && DISPLAYSCREEN)
            printf ("[auvordersfile = %x, opened succesfully]\n",
                      auvordersfile);

        fprintf (auvordersfile,
"\n\n");
        fprintf (auvordersfile,
"# NPS AUV file %s: commanded propulsion orders versus time\n",
                 AUVORDERSFILENAME);
        fprintf (auvordersfile,
"#\n");
        fprintf (auvordersfile,
"#        timestep:  %4.2f seconds\n", dt);
        fprintf (auvordersfile,
"#\n");
        fprintf (auvordersfile,
"# time  heading North East Depth    rpm   rpm    stern  stern   vertical
lateral \n");
        fprintf (auvordersfile,
"#                 x     y     z     port  stbd   plane  rudder  thrusters
thrusters\n");
        fprintf (auvordersfile,
"#                                                               bow/stern
bow/stern\n");
        fprintf (auvordersfile,
"\n");
     }
     else if (TRACE && DISPLAYSCREEN)
            printf ("[auvscriptfile checks out as ready...]\n");

     read_another_line = TRUE;
```

```
while (read_another_line == TRUE)    /* ********* Parse loop *********** */
{
   parameter1 = 0.0;
   parameter2 = 0.0;
   parameter3 = 0.0;

   if (KEYBOARDINPUT == TRUE)
   {
          strcpy (buffer, "Enter mission script command");
          send_data_on_virtual_world_socket (); /* buffer msg sent */
          printf ("\n%s *** HERE ***:   ", buffer);
          gets (command_buffer);
   }
   else
   {
       strcpy (command_buffer, "");
       fgets  (command_buffer, 120, auvscriptfile);

       if (feof (auvscriptfile))
       {
           if (DISPLAYSCREEN)
printf ("[EOF condition:  (%s copy) mission.script.backup, file closed]\n",
                                            AUVSCRIPTFILENAME);
           fclose (auvscriptfile);
           auvscriptfilequit = TRUE;
           read_another_line = FALSE;
           end_test          = TRUE;
           strcpy (command_buffer, "");
           break;
       }
   }

   if ((int)(strlen (command_buffer) <= 120) && TRACE && DISPLAYSCREEN)
   {
       printf ("strlen (command_buffer) = %d", strlen (command_buffer));
       printf (">>>%s<<<", command_buffer);
   }

   parameters_read = sscanf (command_buffer, "%s", keyword);

   if (TRACE && DISPLAYSCREEN)
   {
       printf ("parameters_read = %d, keyword = %s",
                parameters_read,       keyword);
   }

   for (index=0; index<=(int)strlen (keyword); index++) /* set uppercase */
         keyword [index] = toupper (keyword [index]);

   audible_command = TRUE;

   if (TRACE && DISPLAYSCREEN)
   {
       printf (", uppercase  keyword = %s\n",  keyword);
   }

   if      ((parameters_read != 1)          ||
            (strlen (keyword)          == 0) ||
            (strlen (command_buffer)   == 0) ||
            (command_buffer [0]        == '\n'))            /* blank line */
   {
          audible_command = FALSE;
          printf ("\n");
   }
   else    if  ( keyword [0] == '#')                          /* comment */
   {
          if (DISPLAYSCREEN)  printf ("%s", command_buffer);
```

```
        command_buffer [0] = ' ';
    }
    else    if (((keyword [0] == '/') && (keyword [1] == '/')) ||
                ((keyword [0] == '/') && (keyword [1] == '*'))) /* comment */
    {
        if (DISPLAYSCREEN)  printf ("%s", command_buffer);
        command_buffer [0] = ' ';
        command_buffer [1] = ' ';
    }
    else if ((strcmp (keyword,  "HELP") == 0) ||
             (strcmp (keyword,  "?")    == 0) ||
             (strcmp (keyword,  "-?")   == 0) ||
             (strcmp (keyword,  "/?")   == 0))
    {
        if (DISPLAYSCREEN) printf ("[HELP] ");
        print_valid_keywords ();
    }
    else    if      ((strcmp (keyword,  "WAIT") == 0) ||
                      (strcmp (keyword,  "RUN")  == 0))
    {
        parameters_read = sscanf (command_buffer, "%s%lf",
                                               keyword, & parameter1);
        printf ("[%s     %6.2f; ", keyword, parameter1);
        if  ((parameters_read == 2) && (parameter1 >= 0.0))
        {
            read_another_line = FALSE;
            time_next_command = t + parameter1;
            printf ("time of next command %6.2f]\n",
                      time_next_command);
            fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                      t, psi_command, x_command, y_command, z_command,
                      port_rpm_command, stbd_rpm_command,
                      rudder_command, planes_command,
                        bow_lateral_thruster_command,
                      stern_lateral_thruster_command,
                        bow_vertical_thruster_command,
                      stern_vertical_thruster_command);
        }
        else printf (" warning: illegal time value, ignored\n");
    }
    else    if ((strcmp (keyword,  "TIME")       == 0) ||
                (strcmp (keyword,  "WAITUNTIL")  == 0) ||
                (strcmp (keyword,  "PAUSEUNTIL") == 0))
    {
        parameters_read = sscanf (command_buffer, "%s%lf",
                                               keyword, & parameter1);
        printf ("[%s     %6.2f]\n", keyword, parameter1);
        if  (parameters_read == 2)
        {
            read_another_line = FALSE;
            time_next_command = parameter1;
            fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                      t, psi_command, x_command, y_command, z_command,
                      port_rpm_command, stbd_rpm_command,
                      rudder_command, planes_command,
                        bow_lateral_thruster_command,
                      stern_lateral_thruster_command,
                        bow_vertical_thruster_command,
                      stern_vertical_thruster_command);
            if  (parameter1 <= t)
            {
                t = parameter1;
                printf (" warning: time value has reset AUV clock.\n");
```

```
                    read_another_line = TRUE;   /* no PDU */
                }
            }
            else printf (" warning: illegal time value, ignored.\n");
        }
        else if ((strcmp (keyword, "TIMESTEP")  == 0) ||
                 (strcmp (keyword, "TIME-STEP") == 0))
        {
            if (sscanf (command_buffer, "%s%F", keyword, &parameter1) == 2)
            {
                if ((parameter1 > 0.0) && (parameter1 <= 5.0))
                {
                    dt = parameter1;
                    if (DISPLAYSCREEN)
                        printf ("[TIMESTEP   %6.2f] ", dt);
                    fprintf (auvordersfile, "# timestep:  %4.2f seconds\n", dt);
                }
                else print_help = TRUE;
            }
            else print_help = TRUE;
        }
        else if ((strcmp (keyword,  "PAUSE") == 0)    ||
                 (strcmp (keyword,  "-PAUSE") == 0))
        {
            if (DISPLAYSCREEN)
            {
                printf ("[PAUSE]\n");
                strcpy (buffer, " Press any key to continue");
                send_data_on_virtual_world_socket (); /* buffer msg sent */
                printf ("\n%s *** HERE ***:  ", buffer);
                answer = getchar (); /* pause */
            }
        }
        else if ((strcmp (keyword,  "REALTIME")  == 0) ||
                 (strcmp (keyword,  "REAL-TIME") == 0))
        {
                if (DISPLAYSCREEN) printf ("[REALTIME] ");
                REALTIME = TRUE;
        }
        else if ((strcmp (keyword,  "MISSION")   == 0) ||
                 (strcmp (keyword,  "SCRIPT")    == 0) ||
                 (strcmp (keyword,  "FILENAME")  == 0))
        {
                parameters_read = sscanf (command_buffer, "%s%s",
                                                     keyword, new_filename);
                if  (parameters_read == 2)
                {
                    if (DISPLAYSCREEN)
                        printf ("[%s   %s]\n",     keyword, new_filename);
#if defined (sgi)
                     sprintf (backupcommand, "cp   %s %s", new_filename,
                              AUVSCRIPTFILENAME);
#else
                     sprintf (backupcommand, "copy %s %s", new_filename,
                              AUVSCRIPTFILENAME);
#endif
                     if (DISPLAYSCREEN)
                       printf ("%s\n", backupcommand);
                     system    (         backupcommand);
                }
                else
                {
                     if (DISPLAYSCREEN)
              printf ("[%s]  warning: no filename present, ignored\n", keyword);
                }
        }
        else if ((strcmp (keyword,  "KEYBOARD")       == 0)  ||
```

```
                (strcmp (keyword,  "KEYBOARD-ON")    == 0) ||
                (strcmp (keyword,  "KEYBOARD-INPUT") == 0) ||
                (strcmp (keyword,  "KEYBOARDINPUT")  == 0))
        {
               if (DISPLAYSCREEN) printf ("[%s]\n", keyword);
               KEYBOARDINPUT = TRUE;
        }
        else if ((strcmp (keyword,  "KEYBOARD-OFF") == 0) ||
                (strcmp (keyword,  "NO-KEYBOARD")  == 0))
        {
               if (DISPLAYSCREEN) printf ("[%s]\n", keyword);
               KEYBOARDINPUT = FALSE;
        }
        else if ((strcmp (keyword,  "NOWAIT")      == 0) ||
                (strcmp (keyword,  "NO-WAIT")      == 0) ||
                (strcmp (keyword,  "NOREALTIME")   == 0) ||
                (strcmp (keyword,  "NO-REALTIME")  == 0) ||
                (strcmp (keyword,  "NONREALTIME")  == 0) ||
                (strcmp (keyword,  "NO-PAUSE")     == 0) ||
                (strcmp (keyword,  "NOPAUSE")      == 0))
        {
               if (DISPLAYSCREEN) printf ("[%s]\n", keyword);
               REALTIME = FALSE;
        }
        else   if     ((strcmp (keyword,  "QUIT")     == 0) ||
                       (strcmp (keyword,  "STOP")     == 0) ||
                       (strcmp (keyword,  "DONE")     == 0) ||
                       (strcmp (keyword,  "EXIT")     == 0) ||
                       (strcmp (keyword,  "REPEAT")   == 0) ||
                       (strcmp (keyword,  "RESTART")  == 0) ||
                       (strcmp (keyword,  "COMPLETE") == 0) ||

                       (strcmp (keyword,  "KILL")     == 0) ||
                       (strcmp (keyword,  "SHUTDOWN") == 0))
        {
               /* note this command does not reset LOOPFOREVER, except for    */
               /* KILL/SHUTDOWN which terminate the dynamics model connection */
               if ((strcmp (keyword,  "KILL")     == 0) ||
                   (strcmp (keyword,  "SHUTDOWN") == 0))
               {
                  LOOPFOREVER = FALSE;
                  strcpy (buffer, "KILL");
                  send_data_on_virtual_world_socket (); /* buffer msg sent */
               }
               printf ("[%s]\n", keyword);
               if (DISPLAYSCREEN) printf ("[end_test set TRUE]\n");
               end_test          = TRUE;
               read_another_line = FALSE;

               fclose (auvscriptfile);
               auvscriptfilequit = TRUE;
               if (DISPLAYSCREEN)
printf("[QUIT condition: (%s backup file) mission.script.backup, file closed]\n",
                                     AUVSCRIPTFILENAME);
               fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                             t, psi_command, x_command, y_command, z_command,
                            port_rpm_command, stbd_rpm_command,
                              rudder_command,   planes_command,
                                 bow_lateral_thruster_command,
                               stern_lateral_thruster_command,
                                bow_vertical_thruster_command,
                              stern_vertical_thruster_command);
               x_dot                  = 0.0;
               y_dot                  = 0.0;
               z_dot                  = 0.0;
```

```
phi_dot                    = 0.0; /* degrees/sec */
theta_dot                  = 0.0; /* degrees/sec */
psi_dot                    = 0.0; /* degrees/sec */
speed                      = 0.0;
u                          = 0.0;
v                          = 0.0;
w                          = 0.0;
p                          = 0.0; /* degrees/sec */
q                          = 0.0; /* degrees/sec */
r                          = 0.0; /* degrees/sec */
delta_planes               = 0.0; /* degrees     */
delta_rudder               = 0.0; /* degrees     */
port_rpm                   = 0;
stbd_rpm                   = 0;
vertical_thruster_volts = 0.0;
 lateral_thruster_volts = 0.0;

}
else    if ((strcmp (keyword,  "RPM")          == 0) ||
            (strcmp (keyword,  "SPEED")        == 0) ||
            (strcmp (keyword,  "PROPS")        == 0) ||
            (strcmp (keyword,  "PROPELLORS") == 0))
{
        parameters_read = sscanf (command_buffer, "%s%lf%lf",
                                              keyword,     & parameter1,
                                            & parameter2);
        if  (parameters_read == 3)
        {
             printf ("[%s   %6.2f %6.2f]\n",    keyword,      parameter1,
                                              parameter2);
             port_rpm_command    = parameter1;
             stbd_rpm_command    = parameter2;
        }
        else
        {
           parameters_read = sscanf (command_buffer, "%s%lf",
                                                 keyword, & parameter1);
           printf ("[%s       %6.2f]\n", keyword, parameter1);
           if  (parameters_read == 2)
           {
                port_rpm_command    = parameter1;
                stbd_rpm_command    = parameter1;
           }
           else printf (" warning: no value, ignored\n");
        }
}
else    if ((strcmp (keyword,  "COURSE")  == 0) ||
            (strcmp (keyword,  "HEADING") == 0) ||
            (strcmp (keyword,  "YAW")     == 0))
{
        parameters_read = sscanf (command_buffer, "%s%lf",
                                                keyword, & parameter1);
        printf ("[%s   %6.2f]\n", keyword, parameter1);
        if  (parameters_read == 2)
        {
             DEADSTICKRUDDER = FALSE;
             WAYPOINTCONTROL = FALSE;
             psi_command      = parameter1;
              rotate_command = 0.0;
             lateral_command = 0.0;
             ROTATECONTROL   = FALSE;
             LATERALCONTROL  = FALSE;
        }
        else printf (" warning: no value, ignored\n");
}
else    if ((strcmp (keyword,  "TURN")              == 0) ||
            (strcmp (keyword,  "CHANGE-COURSE") == 0))
```

```
{
      parameters_read = sscanf (command_buffer, "%s%lf",
                                                   keyword, & parameter1);
      printf ("[%s     %6.2f]\n", keyword, parameter1);
      if  (parameters_read == 2)
      {
          DEADSTICKRUDDER = FALSE;
          WAYPOINTCONTROL = FALSE;
          ROTATECONTROL   = FALSE;
          psi_command     = psi_command + parameter1;
      }
      else printf (" warning: no value, ignored\n");
}
else  if (strcmp (keyword,  "RUDDER")          == 0)
{
      parameters_read = sscanf (command_buffer, "%s%lf",
                                                   keyword, & parameter1);
      printf ("[%s   %6.2f]\n", keyword, parameter1);
      if  (parameters_read == 2)
      {
          DEADSTICKRUDDER = TRUE;
          WAYPOINTCONTROL = FALSE;
          ROTATECONTROL   = FALSE;
          HOVERCONTROL    = FALSE;
          rudder_command  = parameter1;
      }
      else
      {
          printf (" warning: improper value, rudder order ignored\n");
      }
}
else  if (strcmp (keyword,  "DEADSTICKRUDDER") == 0)
{
      parameters_read = sscanf (command_buffer, "%s%lf",
                                                   keyword, & parameter1);
      if  (parameters_read == 2)
      {
          printf ("[%s   %6.2f]\n", keyword, parameter1);
          DEADSTICKRUDDER = TRUE;
          WAYPOINTCONTROL = FALSE;
          ROTATECONTROL   = FALSE;
          HOVERCONTROL    = FALSE;
          rudder_command  = parameter1;
      }
      else
      {
          printf ("[%s]   ", keyword);
          DEADSTICKRUDDER = TRUE;
          WAYPOINTCONTROL = FALSE;
          ROTATECONTROL   = FALSE;
          rudder_command  = 0.0;
          printf (" warning: improper value, rudder set to 0\n");
      }
}
else  if  (strcmp (keyword,  "DEPTH") == 0)
{
      parameters_read = sscanf (command_buffer, "%s%lf",
                                                   keyword, & parameter1);
      printf ("[%s    %6.2f]\n", keyword, parameter1);
      if  (parameters_read == 2)
      {
          DEADSTICKPLANES = FALSE;
          z_command  = parameter1;
      }
      else printf (" warning: no value, ignored\n");
}
else  if (strcmp (keyword,  "PLANES")          == 0)
```

```
{
        parameters_read = sscanf (command_buffer, "%s%lf",
                                                      keyword, & parameter1);
        printf ("[%s   %6.2f]\n", keyword, parameter1);
        if  (parameters_read == 2)
        {
             DEADSTICKPLANES = TRUE;
             planes_command  = parameter1;
        }
        else printf (" warning: improper value, planes order ignored\n");
}
else    if (strcmp (keyword,  "DEADSTICKPLANES") == 0)
{
        parameters_read = sscanf (command_buffer, "%s%lf",
                                                      keyword, & parameter1);
        if  (parameters_read == 2)
        {
             printf ("[%s   %6.2f]\n", keyword, parameter1);
             DEADSTICKPLANES = TRUE;
             planes_command  = parameter1;
        }
        else
        {
             printf ("[%s]  ", keyword);
             DEADSTICKPLANES = TRUE;
             planes_command  = 0.0;
             printf (" warning: improper value, planes set to 0\n");
        }
}
else    if ((strcmp (keyword,  "THRUSTERS-ON") == 0) ||
            (strcmp (keyword,  "THRUSTERS")    == 0) ||
            (strcmp (keyword,  "THRUSTERON")   == 0) ||
            (strcmp (keyword,  "THRUSTERSON")  == 0))
{
        printf ("[%s]\n", keyword);
        THRUSTERCONTROL = TRUE;
}
else    if ((strcmp (keyword,  "NOTHRUSTER")    == 0) ||
            (strcmp (keyword,  "NOTHRUSTERS")   == 0) ||
            (strcmp (keyword,  "THRUSTERS-OFF") == 0) ||
            (strcmp (keyword,  "THRUSTERSOFF")  == 0))
{
        printf ("[%s]\n", keyword);
        THRUSTERCONTROL = FALSE;
}
else    if (strcmp (keyword,  "ROTATE")    == 0)
{
        parameters_read = sscanf (command_buffer, "%s%lf",
                                                      keyword, & parameter1);
        printf ("[%s   %6.2f]\n", keyword, parameter1);
        if  (parameters_read == 2)
        {
             THRUSTERCONTROL = TRUE;
             WAYPOINTCONTROL = FALSE;
             HOVERCONTROL    = FALSE;
              rotate_command = parameter1;
             clamp (&rotate_command, -12.0, 12.0, "rotate_command");
             lateral_command = 0.0;
             ROTATECONTROL   = TRUE;
             LATERALCONTROL  = FALSE;
        }
        else printf (" warning: no value, ignored\n");
}
else    if ((strcmp (keyword,  "NOROTATE")   == 0) ||
            (strcmp (keyword,  "ROTATEOFF")  == 0) ||
            (strcmp (keyword,  "ROTATE-OFF") == 0))
{
```

```
printf ("[%s]\n", keyword);
rotate_command = 0.0;
ROTATECONTROL  = FALSE;
}
else    if (strcmp (keyword,  "LATERAL")    == 0)
{
        parameters_read = sscanf (command_buffer, "%s%lf",
                                              keyword, & parameter1);
        printf ("[%s   %6.2f]\n", keyword, parameter1);
        if  (parameters_read == 2)
        {
             THRUSTERCONTROL = TRUE;
             WAYPOINTCONTROL = FALSE;
             HOVERCONTROL    = FALSE;
              rotate_command = 0.0;
             lateral_command = parameter1;
             clamp (& lateral_command, -0.5, 0.5, "lateral_command");
             ROTATECONTROL   = FALSE;
             LATERALCONTROL  = TRUE;
        }
        else printf (" warning: no value, ignored\n");
}
else    if ((strcmp (keyword,  "NOLATERAL")    == 0) ||
            (strcmp (keyword,  "LATERALOFF")   == 0) ||
            (strcmp (keyword,  "LATERAL-OFF")  == 0))
{
        printf ("[%s]\n", keyword);
        lateral_command = 0.0;
        LATERALCONTROL  = FALSE;
}
else    if ((strcmp (keyword,  "POSITION")    == 0) ||
            (strcmp (keyword,  "LOCATION")    == 0) )
{
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                           keyword,     & parameter1,
                                         & parameter2, & parameter3);
        printf ("[%s   %6.2f %6.2f %6.2f]\n",  keyword,      parameter1,
                                           parameter2,    parameter3);
        if  (parameters_read == 4)
        {
             x = parameter1;
             y = parameter2;
             z = parameter3;
          /* skip line in telemetry file to break point-to-point lines */
             fprintf (auvdatafile, "\n");
        }
        else printf (" warning: invalid x/y/z position, ignored\n");
}
else    if ((strcmp (keyword,  "ORIENTATION") == 0) ||
            (strcmp (keyword,  "ROTATION")    == 0) )
{
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                           keyword,     & parameter1,
                                         & parameter2, & parameter3);
        printf ("[%s   %6.2f %6.2f %6.2f]\n",  keyword,      parameter1,
                                           parameter2,    parameter3);
        if  (parameters_read == 4)
        {
             phi   = parameter1;
             theta = parameter2;
             psi   = parameter3;
        }
        else
        printf (" warning: invalid phi/theta/psi orientation, ignored\n");
}
else    if ((strcmp (keyword,  "CONTINUE") == 0) ||
            (strcmp (keyword,        "GO") == 0))
```

```
{
        if (DISPLAYSCREEN) printf ("[%s]", keyword);
        return; /* no action required */
}
else    if ((strcmp (keyword,         "STEP") == 0) ||
            (strcmp (keyword, "SINGLE-STEP") == 0))
{
        if (DISPLAYSCREEN) printf ("[%s]", keyword);
        time_next_command = t + dt;
        read_another_line = FALSE;
}
else    if ((strcmp (keyword,  "TRACE")    == 0) ||
            (strcmp (keyword,  "TRACE-ON") == 0))
{
        if (DISPLAYSCREEN) printf ("[TRACE = TRUE] ");
        TRACE = TRUE;
}
else if ((strcmp (keyword,  "TRACEOFF")  == 0) ||
         (strcmp (keyword,  "TRACE-OFF") == 0) ||
         (strcmp (keyword,  "NOTRACE")   == 0) ||
         (strcmp (keyword,  "NO-TRACE")  == 0))
{
        if (DISPLAYSCREEN) printf ("[TRACE = FALSE] ");
        TRACE = FALSE;
}
else if ((strcmp (keyword, "LOOPFOREVER")  == 0) ||
         (strcmp (keyword, "LOOP-FOREVER") == 0))
{
        if (DISPLAYSCREEN) printf ("[LOOPFOREVER] ");
        LOOPFOREVER = TRUE;
}
else if ((strcmp (keyword,  "LOOPONCE")  == 0) ||
         (strcmp (keyword,  "LOOP-ONCE") == 0))
{
        if (DISPLAYSCREEN) printf ("[LOOPONCE] ");
        LOOPFOREVER = FALSE;
}
else if ((strcmp (keyword,  "LOOPFILEBACKUP") == 0) ||
         (strcmp (keyword,  "LOOP-FILE-BACKUP") == 0))
{
        if (DISPLAYSCREEN) printf ("[LOOPFILEBACKUP] ");
        LOOPFILEBACKUP = TRUE;
}
else if ((strcmp (keyword,  "ENTERCONTROLCONSTANTS")    == 0) ||
         (strcmp (keyword,  "ENTER-CONTROL-CONSTANTS") == 0))
{
        if (DISPLAYSCREEN) printf ("[ENTERCONTROLCONSTANTS] ");
        ENTERCONTROLCONSTANTS = TRUE;
        get_control_constants ();
        fflush (stdin);
}
else    if ((strcmp (keyword,  "SLIDINGMODECOURSE")   == 0) ||
            (strcmp (keyword,  "SLIDING-MODE-COURSE") == 0))
{
        printf ("[%s = TRUE]\n", keyword);
        SLIDINGMODECOURSE = TRUE;
        WAYPOINTCONTROL = FALSE;
        ROTATECONTROL   = FALSE;
        HOVERCONTROL    = FALSE;
}
else    if ((strcmp (keyword,  "SLIDINGMODEOFF")   == 0) ||
            (strcmp (keyword,  "SLIDING-MODE-OFF") == 0))
{
        printf ("[%s:  SLIDINGMODECOURSE = FALSE]\n", keyword);
        SLIDINGMODECOURSE = FALSE;
}
else    if ((strcmp (keyword,  "TACTICAL")    == 0) ||
```

```
            (strcmp (keyword,  "TACTICAL-ON") == 0))
{
      printf ("[%s]\n", keyword);
      TACTICAL = TRUE;
}
else  if ((strcmp (keyword,  "NO-TACTICAL")  == 0) ||
            (strcmp (keyword,  "TACTICAL-OFF") == 0))
{
      printf ("[%s]\n", keyword);
      TACTICAL = FALSE;
}
else if (strcmp (keyword,  "SONARTRACE") == 0)
{
      if (DISPLAYSCREEN) printf ("[SONARTRACE] ");
      SONARTRACE = TRUE;
}
else if (strcmp (keyword,  "SONARTRACEOFF") == 0)
{
      if (DISPLAYSCREEN) printf ("[SONARTRACEOFF] ");
      SONARTRACE = FALSE;
}
else if (strcmp (keyword,  "SONARINSTALLED") == 0)
{
      if (DISPLAYSCREEN) printf ("[SONARINSTALLED] ");
      SONARINSTALLED = TRUE;
}
else if (strcmp (keyword,  "PARALLELPORTTRACE") == 0)
{
      if (DISPLAYSCREEN) printf ("[PARALLELPORTTRACE] ");
      PARALLELPORTTRACE = TRUE;
}
else if ((strcmp (keyword,  "AUDIBLE")  == 0) ||
          (strcmp (keyword,  "AUDIO")    == 0) ||
          (strcmp (keyword,  "AUDIO-ON") == 0) ||
          (strcmp (keyword,  "SOUND-ON") == 0) ||
          (strcmp (keyword,  "SOUNDON")  == 0) ||
          (strcmp (keyword,  "SOUND")    == 0))
{
      if (DISPLAYSCREEN) printf ("[AUDIBLE] ");
      strcpy (buffer, "AUDIBLE"); /* copy current command to buffer */
      send_data_on_virtual_world_socket (); /* send to sound driver */
}
else if ((strcmp (keyword,  "SILENT")    == 0) ||
          (strcmp (keyword,  "SILENCE")   == 0) ||
          (strcmp (keyword,  "NOSOUND")   == 0) ||
          (strcmp (keyword,  "SOUNDOFF")  == 0) ||
          (strcmp (keyword, "SOUND-OFF") == 0) ||
          (strcmp (keyword,  "AUDIOOFF")  == 0) ||
          (strcmp (keyword, "AUDIO-OFF") == 0) ||
          (strcmp (keyword,  "QUIET")     == 0))
{
      if (DISPLAYSCREEN) printf ("[SILENT] ");
      strcpy (buffer, "SILENT");  /* copy current command to buffer */
      send_data_on_virtual_world_socket (); /* send to sound driver */
}
else if ((strcmp (keyword, "EMAIL")      == 0) ||
          (strcmp (keyword, "EMAIL-ON")   == 0) ||
          (strcmp (keyword, "E-MAIL")     == 0) ||
          (strcmp (keyword, "E-MAIL-ON") == 0) ||
          (strcmp (keyword, "EMAILON")    == 0))
{
      if (TRACE && DISPLAYSCREEN) printf ("[EMAIL ON] ");
      EMAIL = TRUE;
}
else if ((strcmp (keyword, "EMAILOFF")   == 0) ||
          (strcmp (keyword, "EMAIL-OFF")  == 0) ||
          (strcmp (keyword, "E-MAILOFF")  == 0) ||
```

```
                (strcmp (keyword, "E-MAIL-OFF") == 0) ||
                (strcmp (keyword, "NO-E-MAIL")  == 0) ||
                (strcmp (keyword, "NO-EMAIL")   == 0) ||
                (strcmp (keyword, "NO-E-MAIL")  == 0) ||
                (strcmp (keyword, "NOEMAIL")    == 0))
        {
             if (TRACE && DISPLAYSCREEN) printf ("[EMAIL OFF] ");
             EMAIL = FALSE;
        }
        else if ((strcmp (keyword, "WAYPOINT")     == 0) ||
                (strcmp (keyword, "WAYPOINT-ON") == 0))
        {
             parameters_read = sscanf (command_buffer, "%s%lf%lf%lf",
                                                    keyword,    & parameter1,
                                                  & parameter2, & parameter3);
             if  (parameters_read == 4)
             {
                  printf ("[%s   %6.2f %6.2f %6.2f]\n", keyword, parameter1,
                                                    parameter2, parameter3);
                  WAYPOINTCONTROL = TRUE;
                  x_command       = parameter1;
                  y_command       = parameter2;
                  z_command       = parameter3;
           port_rpm_command       = fabs (port_rpm_command); /* ensure fwd  */
           stbd_rpm_command       = fabs (stbd_rpm_command); /* motion only */
             fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                           t, psi_command, x_command, y_command, z_command,
                         port_rpm_command, stbd_rpm_command,
                           rudder_command,   planes_command,
                               bow_lateral_thruster_command,
                             stern_lateral_thruster_command,
                              bow_vertical_thruster_command,
                            stern_vertical_thruster_command);
             }
             else if  (parameters_read == 3)
             {
                  printf ("[%s   %6.2f %6.2f]\n", keyword, parameter1,
                                                          parameter2);
                  WAYPOINTCONTROL = TRUE;
                  x_command       = parameter1;
                  y_command       = parameter2;
           port_rpm_command       = fabs (port_rpm_command); /* ensure fwd  */
           stbd_rpm_command       = fabs (stbd_rpm_command); /* motion only */
             fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                           t, psi_command, x_command, y_command, z_command,
                         port_rpm_command, stbd_rpm_command,
                           rudder_command,   planes_command,
                               bow_lateral_thruster_command,
                             stern_lateral_thruster_command,
                              bow_vertical_thruster_command,
                            stern_vertical_thruster_command);
             }
             else
             {
                  WAYPOINTCONTROL = FALSE;
                  printf ("\n warning: improper number of values\n waypoint");
                  printf ("set to current position but otherwise ignored\n");
                  x_command       = x;
                  y_command       = y;
                  z_command       = z;
             fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
```

```
                          t, psi_command, x_command, y_command, z_command,
                       port_rpm_command, stbd_rpm_command,
                         rudder_command,   planes_command,
                             bow_lateral_thruster_command,
                           stern_lateral_thruster_command,
                            bow_vertical_thruster_command,
                          stern_vertical_thruster_command);
        }
        if  (FOLLOWWAYPOINTMODE == TRUE)
        {
            if (TRACE) printf ("[FOLLOWWAYPOINTMODE check]");
        /* continue until WAYPOINT reached without further script orders */
            time_next_command = t + 2.0 * dt;
            read_another_line = FALSE;
        }
}
else    if ((strcmp (keyword,  "WAYPOINTFOLLOW")   == 0) ||
            (strcmp (keyword,  "WAYPOINT-FOLLOW") == 0) )
{
        printf ("[%s]\n", keyword);
        FOLLOWWAYPOINTMODE = TRUE;
}
else    if ((strcmp (keyword,  "WAYPOINTFOLLOWOFF")    == 0) ||

            (strcmp (keyword,  "WAYPOINT-FOLLOW-OFF") == 0) )
{
        printf ("[%s]\n", keyword);
        FOLLOWWAYPOINTMODE = FALSE;
}
else if ((strcmp (keyword, "STANDOFF")            == 0) ||
          (strcmp (keyword, "STAND-OFF")           == 0) ||
          (strcmp (keyword, "STANDOFFDISTANCE")    == 0) ||
          (strcmp (keyword, "STANDOFF-DISTANCE")   == 0) ||
          (strcmp (keyword, "STAND-OFF-DISTANCE") == 0))
{
        parameters_read = sscanf (command_buffer, "%s%lf",
                                                  keyword,     & parameter1);
        if  (parameters_read == 2)
        {
            printf ("[%s   %6.2f]\n", keyword, parameter1);
            standoff_distance = parameter1;
        }
        else
        {
            printf ("[%s]\n", keyword);
            printf (" warning: no standoff value provided, ignored");
        }
}
else if ((strcmp (keyword, "HOVER")     == 0) ||
         (strcmp (keyword, "HOVER-ON") == 0))
{
        parameters_read = sscanf (command_buffer, "%s%lf%lf%lf%lf%lf",
                                                  keyword,     & parameter1,
                                                & parameter2, & parameter3,
                                                & parameter4, & parameter5);
        if  (parameters_read == 6)
        {
            printf ("[%s   %6.2f %6.2f %6.2f %6.2f %6.2f]\n",
                                                  keyword, parameter1,
                                                  parameter2, parameter3,
                                                  parameter4, parameter5);
            HOVERCONTROL        = TRUE;
            WAYPOINTCONTROL     = FALSE;
            ROTATECONTROL       = FALSE;
            THRUSTERCONTROL     = TRUE;
            DEADSTICKRUDDER     = TRUE;
            FOLLOWWAYPOINTMODE = FALSE;
```

```
                    rudder_command     = 0.0;
                    x_command          = parameter1;
                    y_command          = parameter2;
                    z_command          = parameter3;
                    psi_command        = parameter4;
                    psi_command_hover  = parameter4;
                    standoff_distance  = parameter5;
               fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                             t, psi_command, x_command, y_command, z_command,
                           port_rpm_command, stbd_rpm_command,
                             rudder_command,   planes_command,
                                 bow_lateral_thruster_command,
                               stern_lateral_thruster_command,
                                bow_vertical_thruster_command,
                              stern_vertical_thruster_command);
               }
               else if  (parameters_read == 5)
               {
                    printf ("[%s   %6.2f %6.2f %6.2f %6.2f]\n",
                                                        keyword, parameter1,
                                                        parameter2, parameter3,
                                                        parameter4);
                    HOVERCONTROL        = TRUE;
                    WAYPOINTCONTROL     = FALSE;
                    ROTATECONTROL       = FALSE;
                    THRUSTERCONTROL     = TRUE;
                    DEADSTICKRUDDER     = TRUE;
                    FOLLOWWAYPOINTMODE = FALSE;
                    rudder_command     = 0.0;
                    x_command          = parameter1;
                    y_command          = parameter2;
                    z_command          = parameter3;
                    psi_command        = parameter4;
                    psi_command_hover  = parameter4;
               fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                             t, psi_command, x_command, y_command, z_command,
                           port_rpm_command, stbd_rpm_command,
                             rudder_command,   planes_command,
                                 bow_lateral_thruster_command,
                               stern_lateral_thruster_command,
                                bow_vertical_thruster_command,
                              stern_vertical_thruster_command);
               }
               else if  (parameters_read == 4)
               {
                    printf ("[%s   %6.2f %6.2f %6.2f]\n", keyword, parameter1,
                                                        parameter2, parameter3);
                    HOVERCONTROL     = TRUE;
                    WAYPOINTCONTROL = FALSE;
                    ROTATECONTROL    = FALSE;
                    THRUSTERCONTROL = TRUE;
                    DEADSTICKRUDDER = TRUE;
                    FOLLOWWAYPOINTMODE = FALSE;
                    rudder_command  = 0.0;
                    x_command        = parameter1;
                    y_command        = parameter2;
                    z_command        = parameter3;
               fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                             t, psi_command, x_command, y_command, z_command,
                           port_rpm_command, stbd_rpm_command,
                             rudder_command,   planes_command,
```

```
                                bow_lateral_thruster_command,
                              stern_lateral_thruster_command,
                               bow_vertical_thruster_command,
                             stern_vertical_thruster_command);
          }
          else if  (parameters_read == 3)
          {
               printf ("[%s   %6.2f %6.2f]\n", keyword, parameter1,
                                                 parameter2);
               HOVERCONTROL      = TRUE;
               WAYPOINTCONTROL = FALSE;
               ROTATECONTROL     = FALSE;
               THRUSTERCONTROL = TRUE;
               DEADSTICKRUDDER = TRUE;
               FOLLOWWAYPOINTMODE = FALSE;
               rudder_command  = 0.0;
               x_command         = parameter1;
               y_command         = parameter2;
          fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                             t, psi_command, x_command, y_command, z_command,
                          port_rpm_command, stbd_rpm_command,
                            rudder_command,   planes_command,
                                bow_lateral_thruster_command,
                              stern_lateral_thruster_command,
                               bow_vertical_thruster_command,
                             stern_vertical_thruster_command);
          }
          else if  (parameters_read == 1)
          {
               printf ("[%s]\n", keyword);
               HOVERCONTROL      = TRUE;
               WAYPOINTCONTROL = FALSE;
               ROTATECONTROL     = FALSE;
               THRUSTERCONTROL = TRUE;
               DEADSTICKRUDDER = TRUE;
               FOLLOWWAYPOINTMODE = FALSE;
               rudder_command  = 0.0;
          fprintf (auvordersfile,
"%6.1f  %6.1f %5.1f %5.1f %5.1f %6.1f %6.1f %6.1f %6.1f  %5.1f %5.1f %5.1f
%5.1f\n",
                             t, psi_command, x_command, y_command, z_command,
                          port_rpm_command, stbd_rpm_command,
                            rudder_command,   planes_command,
                                bow_lateral_thruster_command,
                              stern_lateral_thruster_command,
                               bow_vertical_thruster_command,
                             stern_vertical_thruster_command);
          }
          else
          {
               printf ("[%s]\n", keyword);
               printf (" warning: improper number of values, ignored\n");
          }
     }
     else    parse_mission_string_commands (command_buffer);
             /* check other possibilities */

     if (audible_command == TRUE)
     {
        strcpy (buffer, command_buffer); /* copy current command to buffer */
        send_data_on_virtual_world_socket ();      /* send to sound driver */
     }

     if ((print_help == TRUE) && DISPLAYSCREEN)
     {
```

```
        printf ("%s", command_buffer);
        print_valid_keywords ();

        strcpy (buffer, " is an unknown command"); /* copy msg to buffer   */
        send_data_on_virtual_world_socket ();      /* send to sound driver */
      }
      print_help = FALSE; /* reset value */

   } /* loop until read_another_line is FALSE) */

   if (TRACE && DISPLAYSCREEN)
       printf ("[end parse_mission_script_commands ()]\n");

   return;

} /* end parse_mission_script_commands () */

/*****************************************************************************/

void parse_mission_string_commands (command)
     char * command;
{
      int     index;
      int     number_values = 0;
      char    parameter2 [60];

      if (TRACE && DISPLAYSCREEN)
         printf ("\n[parse_mission_string_commands start]\n");

      number_values = sscanf (command_buffer, "%s", keyword);

      for (index = 0; index <= (int) strlen (keyword); index++)
             keyword [index] = toupper (keyword [index]);

      if       (number_values != 1)
      {
             if (DISPLAYSCREEN) printf (" [no parse word found]\n");
             return;
      }
      if ((strcmp (keyword,  "REMOTEHOST")  == 0) ||
          (strcmp (keyword,  "REMOTE")      == 0) ||
          (strcmp (keyword,  "REMOTE-HOST") == 0) ||
          (strcmp (keyword,  "HOST")        == 0))
      {
           if (sscanf (command, "%s %s", keyword, parameter2) == 2)
             {
                  strcpy (virtual_world_remote_host_name, parameter2);
                  if (DISPLAYSCREEN)  printf ("[REMOTE-HOST %s] ",
                                          virtual_world_remote_host_name);
             }
             else print_help = TRUE;
      }
      else  print_help = TRUE;  /* invalid command line entry parameter found */

      if (TRACE && DISPLAYSCREEN)
          printf ("\n[parse_mission_string_commands complete]\n");

      return;

}/* end parse_mission_string_commands () */

/*****************************************************************************/
```

E. *globals.c* Global Variable Instantiation

```
/*******************************************************************************/
/*
 Program:          globals.c

 Authors:          Don Brutzman

 Revised:          21 October 94

 System:           AUV Gespac 68020/68030, OS-9 version 2.4
 Compiler:         Gespac cc Kernighan & Richie (K&R) C

 Compilation:      ftp>     put globals.c
                   auvsim1> chd execution
      [68020]      auvsim1> make -k2f execution
      [68030]      auvsim1> make      execution

      [Irix ]      fletch>  make execution

 Purpose:          Allows repeated use of global variables global.c via global.h
                   in order to prevent compiler warnings

*/

/*******************************************************************************/

#include "defines.h"

/*******************************************************************************/
/* Program configuration flags                                                 */

int     TRACE                      = 0;   /*  1=trace on,       0=trace off        */
int     DISPLAYSCREEN              = 1;   /*  1=screen on,      0=screen off       */
int     LOCATIONLAB                = 1;   /*  1=graphics lab, 0=actual vehicle     */
int     TACTICAL                   = 0;   /*  1=tactical on,  0=tactical off       */
int     LOOPFOREVER                = 0;   /*  1=repeat execution indefinitely      */
int     LOOPFILEBACKUP             = 1;   /*  1=backup files between replications*/

int     PARALLELPORTTRACE          = 0;   /*  1=trace each char received at port */
int     SONARINSTALLED             = 0;   /*  1=sonar head available for query   */
int     SONARTRACE                 = 0;   /*  1=trace on,       0=trace off        */
int     ENTERCONTROLCONSTANTS      = 0;   /*  1=manual entry, 0=default values     */
int     REALTIME                   = 1;   /*  1=1 second real-time waits, 0=none */

int     DEADRECKON                 = 0;   /*  1=dead reckon navigate, 0=regular   */
int     DEADSTICKRUDDER            = 0;   /*  1=use ordered rudder, 0 = control   */
int     DEADSTICKPLANES            = 0;   /*  1=use ordered planes, 0 = control   */
int     SLIDINGMODECOURSE          = 0;   /*  1=use sliding mode,   0 = control   */

int     THRUSTERCONTROL            = 0;   /*  1=use thrusters, 0=use propellers  */
int     ROTATECONTROL              = 0;   /*  1=use thrusters to rotate in place */
int     LATERALCONTROL             = 0;   /*  1=use thrusters for lateral motion */
int     FOLLOWWAYPOINTMODE         = 0;   /*  1=    go to WAYPOINT without WAITs  */
int     WAYPOINTCONTROL            = 0;   /*  1=    go to WAYPOINT                */
int     HOVERCONTROL               = 0;   /*  1=hover at WAYPOINT                 */

#if defined(sgi)
int     EMAIL                      = 1;   /*  1=send e-mail, 0=don't send e-mail */
#else
int     EMAIL                      = 0;   /* can't send e-mail via OS-9 directly */
#endif

int     NOT_YET_REIMPLEMENTED      = 0;   /*  code in block needs reverification */
```

```
double TIMESTEP                  = 0.1; /*  time of a single closed loop         */
                                        /*  add code to warn if exceeded <<<<  */

int    KEYBOARDINPUT             = 0;   /*  1=read keyboard vice mission file  */

/***************************************************************************/
/* files and paths                                                         */

FILE * auvscriptfile;
FILE * auvordersfile;
FILE * auvdatafile;
FILE * auvtextfile;
FILE * controlconstantsfile;
FILE * emailaddressfile;

/* FILE * serialtestfile;  */

 int   serialpath = 0;

 int   sonarpath  = 0;

/***************************************************************************/
/* Variables and data structures                                           */

/* buffers of full strings for byte transfer to tactical level & disk file    */
/* 'buffer' usually < 256, intentionally oversized in case of overflow error  */

   time_t          system_time              = 0;
   struct tm      *system_tmp               = 0;

   /* partial structure template for the MFI (only interested in PIA for now) */

   struct MFI_PIA
   {
    unsigned short pra;               /* port    register A - data direction A */
    unsigned short cra;               /* control register A                    */
    unsigned short prb;               /* port    register B - data direction B */
    unsigned short crb;               /* control register B                    */
   };

   /*  4 Channels of DAC ADA-1 DAC                                             */
   unsigned char  *dac1_a  = (unsigned char  *) DAC1_ADDR;

   /*  8 Channels of DAC DAC-2B                                                */
   unsigned char  *dac2b_a = (unsigned char  *) DAC2B_ADDR;

    /* 16 Channels of ADC ADA-1                                                */
   unsigned char  *adc1_a  = (unsigned char  *) ADC1_ADDR;

   /* 16 Channels of ADC ADC-2                                                 */
   unsigned short *adc2_a  = (unsigned short *) ADC2_ADDR;

   unsigned short *via0    = (unsigned short *) VIA0_ADDR;

   int             telemetry_records_saved = 0;
   int             mission_leg_counter     = 0;
   int             replication_count       = 1;
   int             end_test                = FALSE;
   int             wrap_count              = 0;
   double          t                       = 0.0;
   double          dt                      = 0.1;
   double          rpm                     = 0.0;
   double          speed_limit             = 0.0;
   double          main_motor_delta1       = 0.0;
   double          main_motor_delta2       = 0.0;
   int             main_motor_volt1        = 512;
   int             main_motor_volt2        = 512;
```

```
double          x                        = 0.0;
double          y                        = 0.0;
double          z                        = 0.0;
double          phi                      = 0.0; /* degrees     */
double          theta                    = 0.0; /* degrees     */
double          psi                      = 0.0; /* degrees     */
double          x_dot                    = 0.0;
double          y_dot                    = 0.0;
double          z_dot                    = 0.0;
double          phi_dot                  = 0.0; /* degrees/sec */
double          theta_dot                = 0.0; /* degrees/sec */
double          psi_dot                  = 0.0; /* degrees/sec */
double          speed                    = 0.0;
double          u                        = 0.0;
double          v                        = 0.0;
double          w                        = 0.0;
double          p                        = 0.0; /* degrees/sec */
double          q                        = 0.0; /* degrees/sec */
double          r                        = 0.0; /* degrees/sec */
double          delta_planes             = 0.0; /* degrees     */
double          delta_rudder             = 0.0; /* degrees     */
double          port_rpm                 = 0;
double          stbd_rpm                 = 0;
double          vertical_thruster_volts = 0.0;
double           lateral_thruster_volts = 0.0;

unsigned short  psi_bit_old              = 0;

double          dg_offset                = 0.0;

double          k_psi                    = 0.0;
double          k_r                      = 0.0;
double          k_v                      = 0.0;

double          k_z                      = 0.0;
double          k_w                      = 0.0;
double          k_theta                  = 0.0;
double          k_q                      = 0.0;

double          k_thruster_psi           = 0.0;
double          k_thruster_r             = 0.0;
double          k_thruster_rotate        = 0.0;
double          k_thruster_lateral       = 0.0;
double          k_thruster_z             = 0.0;
double          k_thruster_w             = 0.0;

double          k_propeller_hover        = 0.0;

double          k_surge_hover            = 0.0;

double          k_thruster_hover         = 0.0;
double          k_sway_hover             = 0.0;

double          k_sigma_r                = 12.0;
double          k_sigma_psi              = 28.87;
double          eta_steering             =  0.1;
double          sigma                    =  0.0;

int             mission_legs_total       = 0;

/*      values initialized in parse_mission_script_commands () */
double          time_next_command        = 0.0;
double                psi_command        = 0.0; /* degrees     */
double                psi_command_hover = 0.0; /* degrees     */
double                  x_command        = 0.0; /* feet        */
double                  y_command        = 0.0; /* feet        */
double                  z_command        = 0.0; /* feet        */
```

```
double          stbd_rpm_command           = 0.0;
double          port_rpm_command           = 0.0;
double            planes_command           = 0.0; /* degrees      */
double            rudder_command           = 0.0; /* degrees      */
double            rotate_command           = 0.0; /* degrees/sec  */
double           lateral_command           = 0.0; /* ft/sec       */

double     bow_lateral_thruster_command = 0.0; /* volts -24..24 */
double   stern_lateral_thruster_command = 0.0; /* volts -24..24 */
double    bow_vertical_thruster_command = 0.0; /* volts -24..24 */
double  stern_vertical_thruster_command = 0.0; /* volts -24..24 */

double          waypoint_distance          = 0.0;
double          waypoint_angle             = 0.0;

double          track_angle                =  0.0;
double          along_track_distance       =  0.0;
double          cross_track_distance       =  0.0;
double          standoff_distance          = 10.0;

double          depth_error;

int             roll_rate_0                = 0;
int             pitch_rate_0               = 0;
int             yaw_rate_0                 = 0;
int             roll_0                     = 0;
int             pitch_0                    = 0;
int             z_val0                     = 0;
int             sw1                        = 0;
int             error                      = 0;
int             range                      = 0;
int             bad_rng                    = 0;
int             bad_updates                = 0;
int             range_index                = 0;
double          range1                     = 0.0;
double          range2                     = 0.0;
double          error1                     = 0.0;
double          error2                     = 0.0;
double          avg_rng                    = 0.0;
int             k_range                    = 0;
int             range_array [5000];

int             pointer                    = NULL;
int             speed_array [11];

int             PortAFlag                  = 0.0;

int             tick                       = 0;
int             curr_tick                  = 0;
int             tick1                      = 0;
int             tick2                      = 0;
int             i                          = 0;

int             mask                       = 0x0000ffff;
long            davedate                   = 0;
long            davetime                   = 0;
double          value                      = 0.0;
short           day                        = 0;
char            answer                     = ' ';
int             start_dwell                = 0;

int             socket_descriptor          = 0;
int             socket_accepted            = 0;
int             socket_stream              = 0;

int             socket_length              = 255;  /* max allowed packet size */
```

```
    int             bytes_received          = 0;
    int             bytes_read              = 0;
    int             bytes_written           = 0;
    int             bytes_left              = 0;
    int             bytes_sent              = 0;

/*  char            buffer_array [FILEBUFFERSIZE][256]; -- not implemented      */

    char            buffer                  [300];
    int             buffer_size             = 0;
    int             buffer_index            = 0;
    int             variables_parsed        = 0;

    char            buffer_received         [300],
                    virtual_world_remote_host_name [60],
                    command_buffer          [300];

    FILE          * netstat_fileptr;

    struct sockaddr_in  server_address;

    struct hostent *server_entity;

    char            keyword        [81];
    char            email_address  [81];

    int             shutdown_signal_received    = FALSE;
    int             virtual_world_socket_opened = FALSE;

    char    * ptr_index;

    int     print_help                  = FALSE;

    double          speed_per_rpm = 2.0 / 700.0 ; /* steady state:
                                                     2.0 feet/sec per 700 rpm   */

    clock_t  previousloopclock       = 0;
    clock_t   currentloopclock       = 0;

    /* +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical */
    double AUV_bow_vertical   = 0.0; /* thruster  rpm                         */
    double AUV_stern_vertical = 0.0; /* thruster  rpm                         */
    double AUV_bow_lateral    = 0.0; /* thruster  rpm                         */
    double AUV_stern_lateral  = 0.0; /* thruster  rpm                         */

    /* warning: do not use leading zero with bearings or else read as octal     */
    double AUV_ST1000_bearing =   0.0;/* ST_1000 conical pencil bearing */
    double AUV_ST1000_range   =  10.0;/* ST_1000 conical pencil range   */
    double AUV_ST1000_strength=  20.0;/* ST_1000 conical pencil strength*/

    double AUV_ST725_bearing  =  90.0;/* ST_725  1 x 24 sector bearing  */
    double AUV_ST725_range    =  20.0;/* ST_725  1 x 24 sector range    */
    double AUV_ST725_strength =  10.0;/* ST_725  1 x 24 sector strength */

    int    audible_command    = TRUE;

    int    auvscriptfilequit  = FALSE;

/*****************************************************************************/
```

F. *globals.h* Global Variable Define File to Permit Multiple References

```
/*******************************************************************************/
/*
 Program:          globals.h

 Authors:          Don Brutzman

 Revised:          28 October 94

 System:           AUV Gespac 68020/68030, OS-9 version 2.4
 Compiler:         Gespac cc Kernighan & Richie (K&R) C

 Compilation:      ftp>     put globals.h
                   auvsim1> chd execution
     [68020]       auvsim1> make -k2f execution
     [68030]       auvsim1> make      execution

     [Irix ]       fletch>  make execution

 Purpose:          Allows repeated use of global variables global.c via global.h
                   in order to prevent compiler warnings

*/

/*******************************************************************************/

#include "defines.h"

/*******************************************************************************/
#if defined(GLOBALS_H)
#else
#define GLOBALS_H
/*******************************************************************************/
/* Program configuration flags                                                 */

extern int    TRACE                    ; /*  1=trace on,      0=trace off          */
extern int    DISPLAYSCREEN            ; /*  1=screen on,     0=screen off         */
extern int    LOCATIONLAB              ; /*  1=graphics lab, 0=actual vehicle      */
extern int    TACTICAL                 ; /*  1=tactical on,  0=tactical off        */
extern int    LOOPFOREVER              ; /*  1=repeat execution indefinitely       */
extern int    LOOPFILEBACKUP           ; /*  1=backup files between replications*/

extern int    PARALLELPORTTRACE        ; /*  1=trace each char received at port */
extern int    SONARINSTALLED           ; /*  1=sonar head available for query    */
extern int    SONARTRACE               ; /*  1=trace on,      0=trace off          */
extern int    ENTERCONTROLCONSTANTS    ; /*  1=manual entry, 0=default values      */
extern int    REALTIME                 ; /*  1=1 second real-time waits, 0=none */

extern int    DEADRECKON               ; /*  1=dead reckon navigate, 0=regular    */
extern int    DEADSTICKRUDDER          ; /*  1=use ordered rudder, 0 = control    */
extern int    DEADSTICKPLANES          ; /*  1=use ordered planes, 0 = control    */
extern int    SLIDINGMODECOURSE        ; /*  1=use sliding mode,   0 = control    */

extern int    THRUSTERCONTROL          ; /*  1=use thrusters, 0=use propellers    */
extern int    ROTATECONTROL            ; /*  1=use thrusters to rotate in place */
extern int    LATERALCONTROL           ; /*  1=use thrusters for lateral motion */
extern int    FOLLOWWAYPOINTMODE       ; /*  1=    go to WAYPOINT without WAITs    */
extern int    WAYPOINTCONTROL          ; /*  1=    go to WAYPOINT                  */
extern int    HOVERCONTROL             ; /*  1=hover at WAYPOINT                  */

#if defined(sgi)
extern int    EMAIL                    ; /*  1=send e-mail, 0=don't send e-mail */
#else
extern int    EMAIL                    ; /* can't send e-mail via OS-9 directly */
```

```
#endif

extern int    NOT_YET_REIMPLEMENTED  ; /*  code in block needs reverification */

extern double TIMESTEP               ; /*  time of a single closed loop        */
                                       /*  add code to warn if exceeded <<<<   */

extern int    KEYBOARDINPUT          ; /*  1=read keyboard vice mission file   */

/***************************************************************************/
/* files and paths                                                         */

extern FILE * auvscriptfile;
extern FILE * auvordersfile;
extern FILE * auvdatafile;
extern FILE * auvtextfile;
extern FILE * controlconstantsfile;
extern FILE * emailaddressfile;

/* FILE * serialtestfile;  */

 extern int   serialpath ;

 extern int   sonarpath  ;

/***************************************************************************/
/* Variables and data structures                                           */

/* buffers of full strings for byte transfer to tactical level & disk file   */
/* 'buffer' usually < 256, intentionally oversized in case of overflow error */

extern  time_t          system_time          ;
extern  struct tm      *system_tmp           ;

   /* partial structure template for the MFI (only interested in PIA for now) */

   struct MFI_PIA
   {
    unsigned short pra;            /* port    register A - data direction A */
    unsigned short cra;            /* control register A                    */
    unsigned short prb;            /* port    register B - data direction B */
    unsigned short crb;            /* control register B                    */
   };

   /*  4 Channels of DAC ADA-1 DAC                                          */
   extern unsigned char  *dac1_a  ;

   /*  8 Channels of DAC DAC-2B                                             */
   extern unsigned char  *dac2b_a ;

    /* 16 Channels of ADC ADA-1                                             */
   extern unsigned char  *adc1_a  ;

   /* 16 Channels of ADC ADC-2                                              */
   extern unsigned short *adc2_a  ;

   extern unsigned short *via0    ;

   extern int             telemetry_records_saved ;
   extern int             mission_leg_counter;
   extern int             replication_count       ;
   extern int             end_test                ;
   extern int             wrap_count              ;
   extern double          t                       ;
   extern double          dt                      ;
   extern double          rpm;
   extern double          speed_limit             ;
```

```
extern double           main_motor_delta1        ;
extern double           main_motor_delta2        ;
extern int              main_motor_volt1         ;
extern int              main_motor_volt2         ;

extern double           x                        ;
extern double           y                        ;
extern double           z                        ;
extern double           phi                      ; /* degrees     */
extern double           theta                    ; /* degrees     */
extern double           psi                      ; /* degrees     */
extern double           x_dot                    ;
extern double           y_dot                    ;
extern double           z_dot                    ;
extern double           phi_dot                  ; /* degrees/sec */
extern double           theta_dot                ; /* degrees/sec */
extern double           psi_dot                  ; /* degrees/sec */
extern double           speed                    ;
extern double           u                        ;
extern double           v                        ;
extern double           w                        ;
extern double           p                        ; /* degrees/sec */
extern double           q                        ; /* degrees/sec */
extern double           r                        ; /* degrees/sec */
extern double           delta_planes             ; /* degrees     */
extern double           delta_rudder             ; /* degrees     */
extern double           port_rpm                 ;
extern double           stbd_rpm                 ;
extern double           vertical_thruster_volts  ;
extern double            lateral_thruster_volts  ;

extern unsigned short   psi_bit_old              ;

extern double           dg_offset                ;

extern double           k_psi                    ;
extern double           k_r                      ;
extern double           k_v                      ;

extern double           k_z                      ;
extern double           k_w                      ;
extern double           k_theta                  ;
extern double           k_q                      ;

extern double           k_thruster_psi           ;
extern double           k_thruster_r             ;
extern double           k_thruster_rotate        ;
extern double           k_thruster_lateral       ;
extern double           k_thruster_z             ;
extern double           k_thruster_w             ;

extern double           k_propeller_hover        ;

extern double           k_surge_hover            ;

extern double           k_thruster_hover         ;
extern double           k_sway_hover             ;

extern double           k_sigma_r                ;
extern double           k_sigma_psi              ;
extern double           eta_steering             ;
extern double           sigma                    ;

extern int              mission_legs_total       ;

/*        values initialized in parse_mission_script_commands () */
extern double           time_next_command        ;
```

```
extern double                  psi_command       ; /* degrees      */
extern double                  psi_command_hover ; /* degrees      */
extern double                    x_command       ; /* feet         */
extern double                    y_command       ; /* feet         */
extern double                    z_command       ; /* feet         */
extern double             stbd_rpm_command       ;
extern double             port_rpm_command       ;
extern double               planes_command       ; /* degrees      */
extern double               rudder_command       ; /* degrees      */
extern double               rotate_command       ; /* degrees/sec  */
extern double              lateral_command       ; /* ft/sec       */

extern double    bow_lateral_thruster_command ; /* volts -24..24 */
extern double  stern_lateral_thruster_command ; /* volts -24..24 */
extern double   bow_vertical_thruster_command ; /* volts -24..24 */
extern double stern_vertical_thruster_command ; /* volts -24..24 */

extern double           waypoint_distance      ;
extern double           waypoint_angle         ;

extern double           track_angle            ;
extern double           along_track_distance   ;
extern double           cross_track_distance   ;
extern double           standoff_distance      ;

extern double           depth_error;

extern int              roll_rate_0            ;
extern int              pitch_rate_0           ;
extern int              yaw_rate_0             ;
extern int              roll_0                 ;
extern int              pitch_0                ;
extern int              z_val0                 ;
extern int              sw1                    ;
extern int              error                  ;
extern int              range                  ;
extern int              bad_rng                ;
extern int              bad_updates            ;
extern int              range_index            ;
extern double           range1                 ;
extern double           range2                 ;
extern double           error1                 ;
extern double           error2                 ;
extern double           avg_rng                ;
extern int              k_range                ;
extern int              range_array [3000];

extern int              pointer                ;
extern int              speed_array [11]       ;

extern int              PortAFlag              ;

extern int              tick                   ;
extern int              curr_tick              ;
extern int              tick1                  ;
extern int              tick2                  ;
extern int              i                      ;

extern int              mask                   ;
extern long             davedate               ;
extern long             davetime               ;
extern double           value                  ;
extern short            day                    ;
extern char             answer                 ;
extern int              start_dwell            ;

extern int              socket_descriptor      ;
```

```
extern int              socket_accepted         ;
extern int              socket_stream           ;

extern int              socket_length           ;/* max allowed packet size */

extern int              bytes_received          ;
extern int              bytes_read              ;
extern int              bytes_written           ;
extern int              bytes_left              ;
extern int              bytes_sent              ;

/* char         buffer_array [FILEBUFFERSIZE][256]; -- not implemented     */

extern char             buffer                  [300];
extern int              buffer_size             ;
extern int              buffer_index            ;
extern int              variables_parsed        ;

extern char             buffer_received         [300],
                    virtual_world_remote_host_name [60],
                        command_buffer          [300];

extern FILE           * netstat_fileptr;

extern struct sockaddr_in  server_address;

extern struct hostent *server_entity;

extern char             keyword       [81];
extern char             email_address [81];

extern int              shutdown_signal_received    ;
extern int              virtual_world_socket_opened ;

extern char     * ptr_index;

extern int        print_help;

extern double     speed_per_rpm; /* steady state:  2.0 feet/sec per 700 rpm  */

extern clock_t  previousloopclock;
extern clock_t   currentloopclock;

/* +- 24V <=> +-2 lb, + Volts moves thruster in + direction, all identical */
extern double AUV_bow_vertical   ;      /* thruster  rpm                   */
extern double AUV_stern_vertical ;      /* thruster  rpm                   */
extern double AUV_bow_lateral    ;      /* thruster  rpm                   */
extern double AUV_stern_lateral  ;      /* thruster  rpm                   */

/* warning: do not use leading zero with bearings or else read as octal */
extern double AUV_ST1000_bearing ;       /* ST_1000 conical pencil bearing */
extern double AUV_ST1000_range   ;       /* ST_1000 conical pencil range   */
extern double AUV_ST1000_strength;       /* ST_1000 conical pencil strength*/

extern double AUV_ST725_bearing  ;       /* ST_725  1 x 24 sector bearing  */
extern double AUV_ST725_range    ;       /* ST_725  1 x 24 sector range    */
extern double AUV_ST725_strength ;       /* ST_725  1 x 24 sector strength */

extern int    audible_command    ;

extern int    auvscriptfilequit  ;

/***************************************************************************/

#endif
```

G. *Makefile.OS9* for Execution Level under OS-9 Operating System

```
# AUV execution level Makefile.OS9

# Don Brutzman  27 OCT 94

# modified OS-9 makefile (originally from /h0/INET/SOURCE/)
# this Makefile is necessary to resolve network libraries compilation

# Note that OS-9 Make syntax is nonstandard.

DD      = /h0
LIB             = -l=$(DD)/lib/sys.l
SOCKLIB         = .../lib/socklib.l
NETLIB  = .../lib/netdb.l
#RDIR   = RELS
RDIR    = .
ODIR    = $(DD)/cmds
RFLAGS          =
CFLAGS          = -bp -g

SOURCE  = recv_str      xmit_str        os9sender      os9server
RSOURCE = recv_str.r    xmit_str.r      os9sender.r    os9server.r

make.date: $(SOURCE)
        touch make.date

globals: $(SOCKLIB. $(NETLIB) globals.c globals.h defines.h Makefile
        cc $(CFLAGS) globals.c -r -l=../$(NETLIB) -l=../$(SOCKLIB) $(LIB)

parse_functions: $(SOCKLIB) $(NETLIB) parse_functions.c globals.h defines.h
Makefile
        cc $(CFLAGS) parse_functions.c -r -l=../$(NETLIB) -l=../$(SOCKLIB) $(LIB)

execution: $(SOCKLIB) $(NETLIB)  globals.r parse_functions.r    execution.c
globals.h defines.h Makefile
        cc $(CFLAGS) execution.c globals.r parse_functions.r -f=execution    \
                -l=../$(NETLIB) -l=../$(SOCKLIB) $(LIB) -lm

# clean:
#       rm *.r /h0/CMDS/execution

# old versions
# -k=0 68000 (default)
# -k=2 68020
# -t=/r0      puts temporary files in ramdisk and speeds compilation
# $*.r        saves relocatable modules in RDIR
# $(SOURCE): $(SOCKLIB) $(NETLIB)
#       cc $(CFLAGS) $*.r -f=$* -l=../$(NETLIB) -l=../$(SOCKLIB) $(LIB)
#       cc $(CFLAGS) execution.c -r    -l=../$(NETLIB) -l=../$(SOCKLIB) $(LIB)
```

H. *Makefile.SGI* for Execution Level under SGI Irix 5.2 Operating System

```
# AUV execution level Makefile.sgi

# Don Brutzman  27 OCT 94

execution: globals.c parse_functions.c execution.c globals.h defines.h Makefile
	cc	-g -lm -cckr -w                            \
		globals.c parse_functions.c execution.c -o execution

# The 'warnings' make option gives voluminous diagnostics which are useful
# in preventing mysterious bugs when the execution code is ported to OS-9.
# Similar to lint.

warnings:
	cc	-g -lm -cckr -fullwarn -wlint,p \
		globals.c parse_functions.c execution.c -o execution


all:
	touch globals.c parse_functions.c execution.c
	make execution


clean:
	rm execution *.o
```

I. *startup* OS-9 Reboot System Configuration File

```
-t -np
*
*   OS-9 startup file
*   Last modification 22 JUN 94     Walt Landaker & Don Brutzman
*
*
*
*
setime -s                       ;* start system clock
link shell cio                  ;* make "shell" and "cio" stay in memory

* the following are already in RAM memory from hard disk bootfile /h0/OS9boot
* iniz r0 h0 d0 t1 p1           ;* initialize devices
* load utils                    ;* make some utilities stay in memory
* load bootobjs/dd.r0           ;* get the default device descriptor
* init.ramdisk >/nil >>/nil&    ;* initialize it if its the ram disk
* tsmon /t1 &                   ;* start other terminals

setenv TERM vt100
startlan
wait
free
list sys/motd
shell<>>>/term -e=/h0/sys/errmsg.short
login
```

J. *.login* OS-9 User Login and Unix Alias File

```
# auvsimi .login file

# for OS-9 with 'sh' shell

# Don Brutzman  22 June 94

history 40
setenv TERM vt100

alias cat      list
alias cd       chd
alias chmod    attr -?
alias clear    cls
alias cp       copy
alias h        history
alias l        dir -ade
alias ls       dir -ad
alias m        list
alias man      help
alias mkdir    makdir
alias more     list
alias lo       logout
alias nopause  tmode nopause
alias pause    tmode pause
alias ps       procs
alias pwd      pd
alias quit     logout
alias reboot   shell break
alias ren      rename
alias rm       del
alias rmdir    deldir
alias s        sh
alias source   sh
alias type     list

# following doesn't seem to work
#   automatically on telnet logins
tmode nopause

echo
echo "telnet users type:  nopause"
```

K. *auv_plot.gnu*: Execution Level Telemetry Plotting using *gnuplot*

```
###########################################################################
#                                                                         #
#        filename:  auv_plot.gnu                                          #
#                                                                         #
#        function:  GNUPLOT V3.5 script to plot AUV telemetry data       #
#                   to screen & to PostScript files                       #
#                                                                         #
#         updated:  18 OCT 94                                             #
#                                                                         #
#          author:  Don Brutzman                                          #
#                                                                         #
#       execution:  gnuplot> load "auv_plot.gnu"                          #
#                   gnuplot> reread                                       #
#                                                                         #
#                   unix> gnuplot  auv_plot.gnu                           #
#                                                                         #
#    re-plotting:                                                         #
#       'xpsview'   -wp -skipc -or landscape ~/execution/AUV_telemetry.ps & #
#       ghostview              -landscape ~/execution/AUV_telemetry.ps &  #
#                                                                         #
# C program call:   system ("gnuplot auv_plot.gnu");                      #
#                                                                         #
#       telemetry:  mission.output.telemetry (AUV telemetry 0.1 sec interval)#
#       alternate:  mission.output.1_second  (AUV telemetry 1.0 sec interval)#
#                                                                         #
# alternate plot:   unix> gnuplot  auv_plot_1_second.gnu                  #
#                                                                         #
#   output files:   AUV_telemetry.ps & *.eps plots                        #
#                                                                         #
#  related files:   execution.c                                           #
#                   underwater virtual world                              #
#                                                                         #
# output archive:   ftp://taurus.cs.nps.navy.mil/pub/auv/AUV_telemetry.ps.Z #
#                                                                         #
#    gnuplot FAQ:     ftp://ftp.dartmouth.edu/pub/gnuplot/faq/gpt_faq.html #
#                                                                         #
#   distribution:     http://www.cs.dartmouth.edu/gnuplot/                #
###########################################################################

# Plot filenames:

# combined  AUV_telemetry.ps
# figure  1 AUV_x_y.eps
# figure  2 AUV_t_x.eps
# figure  3 AUV_t_y.eps
# figure  4 AUV_t_z.eps
# figure  5 AUV_t_phi.eps
# figure  6 AUV_t_theta.eps
# figure  7 AUV_t_theta_all.eps
# figure  8 AUV_t_psi.eps
# figure  9 AUV_t_lateral_thrusters.eps
# figure 10 AUV_t_vertical_thrusters.eps
# figure 11 AUV_t_u.eps
# figure 12 AUV_t_v.eps
# figure 13 AUV_t_w.eps
# figure 14 AUV_t_p.eps
# figure 15 AUV_t_q.eps
# figure 16 AUV_t_r.eps
# figure 17 AUV_t_delta_rudder.eps
# figure 18 AUV_t_delta_planes.eps
# figure 19 AUV_t_rpm.eps
# figure 20 AUV_t_thrusters.eps

###########################################################################

# setup:

set terminal x11

set time
set grid
```

```
set data style linespoints

set samples 10        # data point plotting frequency

##########################################################################

set xlabel 'East -> (y_world) [ft]'
set ylabel 'North ^ (x_world) [ft]'

set title  'NPS AUV telemetry  1' 26,.8
plot  'mission.output.telemetry'        using 4:3  title 'x vs y (geographic position plot)'
pause -1 'hit enter to continue with plot 2 '

set xlabel 'time t (seconds)'
set ylabel

set title  'NPS AUV telemetry  2' 26,.8
plot  'mission.output.telemetry'        using 2:3  title 't vs x      [ft]      ', \
      'mission.output.telemetry'        using 2:15 title 't vs x_dot [ft/sec]'
pause -1 'hit enter to continue with plot 3 '

set title  'NPS AUV telemetry  3' 26,.8
plot  'mission.output.telemetry'        using 2:4  title 't vs y      [ft]      ', \
      'mission.output.telemetry'        using 2:16 title 't vs y_dot [ft/sec]'
pause -1 'hit enter to continue with plot 4 '

set title  'NPS AUV telemetry  4' 26,.8
plot  'mission.output.telemetry'        using 2:5  title 't vs z      (depth)       [ft]      ', \
      'mission.output.telemetry'        using 2:17 title 't vs z_dot (depth rate)  [ft/sec]', \
      'mission.output.telemetry'        using 2:22 title 't vs delta_planes         [deg]     ', \
      'mission.output.orders'           using 1:3  title 't vs ordered depth        [ft]      ' with steps
pause -1 'hit enter to continue with plot 5 '

set title  'NPS AUV telemetry  5' 26,.8
plot  'mission.output.telemetry'        using 2:6  title 't vs phi      (roll: x axis) [deg]      ', \
      'mission.output.telemetry'        using 2:18 title 't vs phi_dot (roll rate)     [deg/sec]'
pause -1 'hit enter to continue with plot 6 '

set title  'NPS AUV telemetry  6' 26,.8
plot  'mission.output.telemetry'        using 2:7  title 't vs theta      (elevation angle) [deg]      ', \
      'mission.output.telemetry'        using 2:19 title 't vs theta_dot (elevation rate)  [deg/sec]'
pause -1 'hit enter to continue with plot 7 '

set title  'NPS AUV telemetry  7' 26,.8
plot  'mission.output.telemetry'        using 2:7  title 't vs theta      (elevation angle) [deg]      ', \
      'mission.output.telemetry'        using 2:19 title 't vs theta_dot (elevation rate)  [deg/sec]', \
      'mission.output.telemetry'        using 2:5  title 't vs z          (depth)           [ft]       ', \
      'mission.output.telemetry'        using 2:11 title 't vs w          (heave)           [ft/sec]  ', \
      'mission.output.telemetry'        using 2:22 title 't vs delta_planes                 [deg]      ', \
      'mission.output.orders'           using 1:3  title 't vs ordered depth                [ft]       ' with
steps
pause -1 'hit enter to continue with plot 8 '

set title  'NPS AUV telemetry  8' 26,.8
plot  'mission.output.telemetry'        using 2:8  title 't vs psi      (azimuth: z axis) [deg]      ', \
      'mission.output.telemetry'        using 2:20 title 't vs psi_dot (azimuth rate)    [deg/sec]', \
      'mission.output.telemetry'        using 2:14 title 't vs r        (yaw rate)        [deg/sec]', \
      'mission.output.telemetry'        using 2:10 title 't vs v        (sway)            [ ft/sec]', \
      'mission.output.telemetry'        using 2:21 title 't vs delta_rudder               [deg]      ', \
      'mission.output.orders'           using 1:2  title 't vs ordered heading            [deg]      ' with steps
pause -1 'hit enter to continue with plot 9 '

set title  'NPS AUV telemetry  9' 26,.8
plot  'mission.output.telemetry'        using 2:8  title 't vs psi      (azimuth: z axis) [deg]      ', \
      'mission.output.telemetry'        using 2:20 title 't vs psi_dot (azimuth rate)    [deg/sec]', \
      'mission.output.telemetry'        using 2:14 title 't vs r        (yaw rate)        [deg/sec]', \
      'mission.output.telemetry'        using 2:10 title 't vs v        (sway)            [ ft/sec]', \
      'mission.output.telemetry'        using 2:27 title 't vs   bow lateral thruster     [volts]   ' with
steps, \
      'mission.output.telemetry'        using 2:28 title 't vs stern lateral thruster     [volts]   ' with
steps, \
      'mission.output.orders'           using 1:2  title 't vs ordered heading            [deg]      ' with steps
pause -1 'hit enter to continue with plot 10 '

set title  'NPS AUV telemetry 10' 26,.8
plot  'mission.output.telemetry'        using 2:5  title 't vs z (depth)                  [ft]      ', \
```

```
      'mission.output.telemetry'      using 2:11 title 't vs w (heave)                     [ft/sec]', \
      'mission.output.telemetry'      using 2:25 title 't vs  bow vertical thruster [volts] ' with steps, \
      'mission.output.telemetry'      using 2:26 title 't vs stern vertical thruster [volts] ' with steps, \
      'mission.output.orders'         using 1:3  title 't vs ordered depth             [ft]    ' with steps
 pause -1 'hit enter to continue with plot 11 '

 set title  'NPS AUV telemetry 11' 26,.8
 plot  'mission.output.telemetry'      using 2:9  title 't vs u (surge) [ft/sec]'
 pause -1 'hit enter to continue with plot 12 '

 set title  'NPS AUV telemetry 12' 26,.8
 plot  'mission.output.telemetry'      using 2:10 title 't vs v (sway) [ft/sec]'
 pause -1 'hit enter to continue with plot 13 '

 set title  'NPS AUV telemetry 13' 26,.8
 plot  'mission.output.telemetry'      using 2:11 title 't vs w (heave) [ft/sec]'
 pause -1 'hit enter to continue with plot 14 '

 set title  'NPS AUV telemetry 14' 26,.8
 plot  'mission.output.telemetry'      using 2:12 title 't vs p (roll rate) [deg/sec]'
 pause -1 'hit enter to continue with plot 15 '

 set title  'NPS AUV telemetry 15' 26,.8
 plot  'mission.output.telemetry'      using 2:13 title 't vs q (pitch rate) [deg/sec]'
 pause -1 'hit enter to continue with plot 16 '

 set title  'NPS AUV telemetry 16' 26,.8
 plot  'mission.output.telemetry'      using 2:14 title 't vs r (yaw rate) [deg/sec]'
 pause -1 'hit enter to continue with plot 17 '

 set title  'NPS AUV telemetry 17' 26,.8
 plot  'mission.output.telemetry'      using 2:21 title 't vs delta_rudder bow [deg]'
 pause -1 'hit enter to continue with plot 18 '

 set title  'NPS AUV telemetry 18' 26,.8
 plot  'mission.output.telemetry'      using 2:22 title 't vs delta_planes bow [deg]'
 pause -1 'hit enter to continue with plot 19 '

 set title  'NPS AUV telemetry 19' 26,.8
 plot  'mission.output.telemetry'      using 2:23 title 't vs rpm_left    [rpm]', \
       'mission.output.telemetry'      using 2:24 title 't vs rpm_right   [rpm]', \
       'mission.output.orders'         using 1:4  title 't vs rpm_ordered [rpm]' with steps
 pause -1 'hit enter to continue with plot 20 '

 set title  'NPS AUV telemetry 20' 26,.8
 plot  'mission.output.telemetry'      using 2:25 title 't vs   bow vertical thruster [volts]' with steps, \
       'mission.output.telemetry'      using 2:26 title 't vs stern vertical thruster [volts]' with steps, \
       'mission.output.telemetry'      using 2:27 title 't vs   bow  lateral thruster [volts]' with steps, \
       'mission.output.telemetry'      using 2:28 title 't vs stern  lateral thruster [volts]' with steps

 pause -1 'hit enter to plot postscript files & quit'

 #########################################################################
 #########################################################################

 pause 0 'plotting combined plot AUV_telemetry.ps ...'

 clear

 # -------------------------------------------------------------------#
 # uncomment to choose one of the following two options:
 # -------------------------------------------------------------------#
 #  60% size for xpsview
 # set size 0.6, 0.6
 # set terminal postscript portrait  color 'Courier' 14
 # set title  'NPS AUV telemetry' 29,.8
 # -------------------------------------------------------------------#
 # full page for printing
   set terminal postscript landscape color 'Courier' 20
   set title  'NPS AUV telemetry' 16,.8
 # -------------------------------------------------------------------#

 set output 'AUV_telemetry.ps'

 pause 0 'page 1'
 set xlabel  'East -> (y_world) [ft]'
```

```
set ylabel "North -> (x_world) [ft]"
set title  "NPS AUV telemetry  1" 14,.8
plot   "mission.output.telemetry"       using 4:3  title "x vs y (geographic position plot)"

set xlabel "time t (seconds)"
set ylabel

pause 0 "page 2"
set title  "NPS AUV telemetry  2" 14,.8
plot   "mission.output.telemetry"       using 2:3  title "t vs x     [ft]      ", \
       "mission.output.telemetry"       using 2:15 title "t vs x_dot [ft/sec]"

pause 0 "page 3"
set title  "NPS AUV telemetry  3" 14,.8
plot   "mission.output.telemetry"       using 2:4  title "t vs y     [ft]      ", \
       "mission.output.telemetry"       using 2:16 title "t vs y_dot [ft/sec]"

pause 0 "page 4"
set title  "NPS AUV telemetry  4" 14,.8
plot   "mission.output.telemetry"       using 2:5  title "t vs z       (depth)        [ft]      ", \
       "mission.output.telemetry"       using 2:17 title "t vs z_dot (depth rate) [ft/sec]", \
       "mission.output.telemetry"       using 2:22 title "t vs delta_planes           [deg]     ", \
       "mission.output.orders"          using 1:3  title "t vs ordered depth          [ft]      " with steps

pause 0 "page 5"
set title  "NPS AUV telemetry  5" 14,.8
plot   "mission.output.telemetry"       using 2:6  title "t vs phi     (roll: x axis) [deg]     ", \
       "mission.output.telemetry"       using 2:18 title "t vs phi_dot (roll rate)    [deg/sec]"

pause 0 "page 6"
set title  "NPS AUV telemetry  6" 14,.8
plot   "mission.output.telemetry"       using 2:7  title "t vs theta     (elevation angle) [deg]     ", \
       "mission.output.telemetry"       using 2:19 title "t vs theta_dot (elevation rate)  [deg/sec]"

pause 0 "page 7"
set title  "NPS AUV telemetry  7" 14,.8
plot   "mission.output.telemetry"       using 2:7  title "t vs theta     (elevation angle) [deg]     ", \
       "mission.output.telemetry"       using 2:19 title "t vs theta_dot (elevation rate)  [deg/sec]", \
       "mission.output.telemetry"       using 2:5  title "t vs z         (depth)           [ft]      ", \
       "mission.output.telemetry"       using 2:11 title "t vs w         (heave)           [ft/sec]  ", \
       "mission.output.telemetry"       using 2:22 title "t vs delta_planes                [deg]     ", \
       "mission.output.orders"          using 1:3  title "t vs ordered depth               [ft]      " with
steps

pause 0 "page 8"
set title  "NPS AUV telemetry  8" 14,.8
plot   "mission.output.telemetry"       using 2:8  title "t vs psi     (azimuth: z axis) [deg]     ", \
       "mission.output.telemetry"       using 2:20 title "t vs psi_dot (azimuth rate)    [deg/sec]", \
       "mission.output.telemetry"       using 2:14 title "t vs r       (yaw rate)        [deg/sec]", \
       "mission.output.telemetry"       using 2:10 title "t vs v       (sway)            [ ft/sec]", \
       "mission.output.telemetry"       using 2:21 title "t vs delta_rudder              [deg]     ", \
       "mission.output.orders"          using 1:2  title "t vs ordered heading           [deg]     " with steps

pause 0 "page 9"
set title  "NPS AUV telemetry  9" 14,.8
plot   "mission.output.telemetry"       using 2:8  title "t vs psi     (azimuth: z axis) [deg]     ", \
       "mission.output.telemetry"       using 2:20 title "t vs psi_dot (azimuth rate)    [deg/sec]", \
       "mission.output.telemetry"       using 2:14 title "t vs r       (yaw rate)        [deg/sec]", \
       "mission.output.telemetry"       using 2:10 title "t vs v       (sway)            [ ft/sec]", \
       "mission.output.telemetry"       using 2:27 title "t vs  bow lateral thruster     [volts]   " with
steps, \
       "mission.output.telemetry"       using 2:28 title "t vs stern lateral thruster    [volts]   " with
steps, \
       "mission.output.orders"          using 1:2  title "t vs ordered heading           [deg]     " with steps

pause 0 "page 10"
set title  "NPS AUV telemetry 10" 14,.8
plot   "mission.output.telemetry"       using 2:5  title "t vs z (depth)                 [ft]      ", \
       "mission.output.telemetry"       using 2:11 title "t vs w (heave)                 [ft/sec]", \
       "mission.output.telemetry"       using 2:25 title "t vs  bow vertical thruster    [volts] " with steps, \
       "mission.output.telemetry"       using 2:26 title "t vs stern vertical thruster   [volts] " with steps, \
       "mission.output.orders"          using 1:3  title "t vs ordered depth             [ft]    " with steps

pause 0 "page 11"
set title  "NPS AUV telemetry 11" 14,.8
plot   "mission.output.telemetry"       using 2:9  title "t vs u (surge) [ft/sec]"
```

```
pause 0 'page 12'
set title  'NPS AUV telemetry 12' 14,.8
plot  'mission.output.telemetry'       using 2:10 title 't vs v (sway) [ft/sec]'

pause 0 'page 13'
set title  'NPS AUV telemetry 13' 14,.8
plot  'mission.output.telemetry'       using 2:11 title 't vs w (heave) [ft/sec]'

pause 0 'page 14'
set title  'NPS AUV telemetry 14' 14,.8
plot  'mission.output.telemetry'       using 2:12 title 't vs p (roll rate) [deg/sec]'

pause 0 'page 15'
set title  'NPS AUV telemetry 15' 14,.8
plot  'mission.output.telemetry'       using 2:13 title 't vs q (pitch rate) [deg/sec]'

pause 0 'page 16'
set title  'NPS AUV telemetry 16' 14,.8
plot  'mission.output.telemetry'       using 2:14 title 't vs r (yaw rate) [deg/sec]'

pause 0 'page 17'
set title  'NPS AUV telemetry 17' 14,.8
plot  'mission.output.telemetry'       using 2:21 title 't vs delta_rudder bow [deg]'

pause 0 'page 18'
set title  'NPS AUV telemetry 18' 14,.8
plot  'mission.output.telemetry'       using 2:22 title 't vs delta_planes bow [deg]'

pause 0 'page 19'
set title  'NPS AUV telemetry 19' 14,.8
plot  'mission.output.telemetry'       using 2:23 title 't vs rpm_left    [rpm]', \
      'mission.output.telemetry'       using 2:24 title 't vs rpm_right   [rpm]', \
      'mission.output.orders'          using 1:4  title 't vs rpm_ordered [rpm]' with steps

pause 0 'page 20'
set title  'NPS AUV telemetry 20' 14,.8
plot  'mission.output.telemetry'       using 2:25 title 't vs   bow vertical thruster [volts]' with steps, \
      'mission.output.telemetry'       using 2:26 title 't vs stern vertical thruster [volts]' with steps, \
      'mission.output.telemetry'       using 2:27 title 't vs   bow  lateral thruster [volts]' with steps, \
      'mission.output.telemetry'       using 2:28 title 't vs stern  lateral thruster [volts]' with steps

##########################################################################
##########################################################################

pause 0 'plotting individual encapsulated postscript files...'

set title  'NPS AUV telemetry' 29,.8

set terminal postscript eps color 'Courier' 14

pause 0 'figure  1 AUV_x_y.eps'
set output        'AUV_x_y.eps'
set xlabel  'East -> (y_world) [ft]'
set ylabel 'North -> (x_world) [ft]'
set title  'NPS AUV telemetry  1' 26,.8
plot  'mission.output.telemetry'       using 4:3  title 'x vs y (geographic position plot)'

set xlabel 'time t (seconds)'
set ylabel

pause 0 'figure  2 AUV_t_x.eps'
set output        'AUV_t_x.eps'
set title  'NPS AUV telemetry  2' 26,.8
plot  'mission.output.telemetry'       using 2:3  title 't vs x     [ft]     ', \
      'mission.output.telemetry'       using 2:15 title 't vs x_dot [ft/sec]'

pause 0 'figure  3 AUV_t_y.eps'
set output        'AUV_t_y.eps'
set title  'NPS AUV telemetry  3' 26,.8
plot  'mission.output.telemetry'       using 2:4  title 't vs y     [ft]     ', \
      'mission.output.telemetry'       using 2:16 title 't vs y_dot [ft/sec]'

pause 0 'figure  4 AUV_t_z.eps'
set output        'AUV_t_z.eps'
set title  'NPS AUV telemetry  4' 26,.8
plot  'mission.output.telemetry'       using 2:5  title 't vs z     (depth)      [ft]     ', \
```

```
     'mission.output.telemetry'      using 2:17 title 't vs z_dot (depth rate) [ft/sec]', \
     'mission.output.telemetry'      using 2:22 title 't vs delta_planes        [deg]    ', \
     'mission.output.orders'         using 1:3  title 't vs ordered depth       [ft]     ' with steps

pause 0 'figure  5 AUV_t_phi.eps'
set output        'AUV_t_phi.eps'
set title  'NPS AUV telemetry  5' 26,.8
plot  'mission.output.telemetry'     using 2:6  title 't vs phi     (roll: x axis) [deg]     ', \
      'mission.output.telemetry'     using 2:18 title 't vs phi_dot (roll rate)    [deg/sec]'

pause 0 'figure  6 AUV_t_theta.eps'
set output        'AUV_t_theta.eps'
set title  'NPS AUV telemetry  6' 26,.8
plot  'mission.output.telemetry'     using 2:7  title 't vs theta     (elevation angle) [deg]     ', \
      'mission.output.telemetry'     using 2:19 title 't vs theta_dot (elevation rate)  [deg/sec]'

pause 0 'figure  7 AUV_t_theta_all.eps'
set output        'AUV_t_theta_all.eps'
set title  'NPS AUV telemetry  7' 26,.8
plot  'mission.output.telemetry'     using 2:7  title 't vs theta     (elevation angle) [deg]     ', \
      'mission.output.telemetry'     using 2:19 title 't vs theta_dot (elevation rate)  [deg/sec]', \
      'mission.output.telemetry'     using 2:5  title 't vs z         (depth)           [ft]      ', \
      'mission.output.telemetry'     using 2:11 title 't vs w         (heave)           [ft/sec]  ', \
      'mission.output.telemetry'     using 2:22 title 't vs delta_planes                [deg]     ', \
      'mission.output.orders'        using 1:3  title 't vs ordered depth               [ft]      ' with
steps

pause 0 'figure  8 AUV_t_psi.eps'
set output        'AUV_t_psi.eps'
set title  'NPS AUV telemetry  8' 26,.8
plot  'mission.output.telemetry'     using 2:8  title 't vs psi     (azimuth: z axis) [deg]     ', \
      'mission.output.telemetry'     using 2:20 title 't vs psi_dot (azimuth rate)    [deg/sec]', \
      'mission.output.telemetry'     using 2:14 title 't vs r       (yaw rate)        [deg/sec]', \
      'mission.output.telemetry'     using 2:10 title 't vs v       (sway)            [ ft/sec]', \
      'mission.output.telemetry'     using 2:21 title 't vs delta_rudder              [deg]     ', \
      'mission.output.orders'        using 1:2  title 't vs ordered heading           [deg]     ' with steps

pause 0 'figure  9 AUV_t_lateral_thrusters.eps'
set output        'AUV_t_lateral_thrusters.eps'
set title  'NPS AUV telemetry  9' 26,.8
plot  'mission.output.telemetry'     using 2:8  title 't vs psi     (azimuth: z axis) [deg]     ', \
      'mission.output.telemetry'     using 2:20 title 't vs psi_dot (azimuth rate)    [deg/sec]', \
      'mission.output.telemetry'     using 2:14 title 't vs r       (yaw rate)        [deg/sec]', \
      'mission.output.telemetry'     using 2:10 title 't vs v       (sway)            [ ft/sec]', \
      'mission.output.telemetry'     using 2:27 title 't vs  bow lateral thruster     [volts]   ' with
steps, \
      'mission.output.telemetry'     using 2:28 title 't vs stern lateral thruster    [volts]   ' with
steps, \
      'mission.output.orders'        using 1:2  title 't vs ordered heading           [deg]     ' with steps

pause 0 'figure 10 AUV_t_vertical_thrusters.eps'
set output        'AUV_t_vertical_thrusters.eps'
set title  'NPS AUV telemetry 10' 26,.8
plot  'mission.output.telemetry'     using 2:5  title 't vs z (depth)                  [ft]     ', \
      'mission.output.telemetry'     using 2:11 title 't vs w (heave)                  [ft/sec]', \
      'mission.output.telemetry'     using 2:25 title 't vs  bow vertical thruster     [volts] ' with steps, \
      'mission.output.telemetry'     using 2:26 title 't vs stern vertical thruster    [volts] ' with steps, \
      'mission.output.orders'        using 1:3  title 't vs ordered depth              [ft]    ' with steps

pause 0 'figure 11 AUV_t_u.eps'
set output        'AUV_t_u.eps'
set title  'NPS AUV telemetry 11' 26,.8
plot  'mission.output.telemetry'     using 2:9  title 't vs u (surge) [ft/sec]'

pause 0 'figure 12 AUV_t_v.eps'
set output        'AUV_t_v.eps'
set title  'NPS AUV telemetry 12' 26,.8
plot  'mission.output.telemetry'     using 2:10 title 't vs v (sway) [ft/sec]'

pause 0 'figure 13 AUV_t_w.eps'
set output        'AUV_t_w.eps'
set title  'NPS AUV telemetry 13' 26,.8
plot  'mission.output.telemetry'     using 2:11 title 't vs w (heave) [ft/sec]'

pause 0 'figure 14 AUV_t_p.eps'
set output        'AUV_t_p.eps'
```

```
set title  'NPS AUV telemetry 14' 26,.8
plot  'mission.output.telemetry'       using 2:12 title 't vs p (roll rate) [deg/sec]'

pause 0 'figure 15 AUV_t_q.eps'
set output          'AUV_t_q.eps'
set title  'NPS AUV telemetry 15' 26,.8
plot  'mission.output.telemetry'       using 2:13 title 't vs q (pitch rate) [deg/sec]'

pause 0 'figure 16 AUV_t_r.eps'
set output          'AUV_t_r.eps'
set title  'NPS AUV telemetry 16' 26,.8
plot  'mission.output.telemetry'       using 2:14 title 't vs r (yaw rate) [deg/sec]'

pause 0 'figure 17 AUV_t_delta_rudder.eps'
set output          'AUV_t_delta_rudder.eps'
set title  'NPS AUV telemetry 17' 26,.8
plot  'mission.output.telemetry'       using 2:21 title 't vs delta_rudder bow [deg]'

pause 0 'figure 18 AUV_t_delta_planes.eps'
set output          'AUV_t_delta_planes.eps'
set title  'NPS AUV telemetry 18' 26,.8
plot  'mission.output.telemetry'       using 2:22 title 't vs delta_planes bow [deg]'

pause 0 'figure 19 AUV_t_rpm.eps'
set output          'AUV_t_rpm.eps'
set title  'NPS AUV telemetry 19' 26,.8
plot  'mission.output.telemetry'       using 2:23 title 't vs rpm_left    [rpm]', \
      'mission.output.telemetry'       using 2:24 title 't vs rpm_right   [rpm]', \
      'mission.output.orders'          using 1:4  title 't vs rpm_ordered [rpm]' with steps

pause 0 'figure 20 AUV_t_thrusters.eps'
set output          'AUV_t_thrusters.eps'
set title  'NPS AUV telemetry 20' 26,.8
plot  'mission.output.telemetry'       using 2:25 title 't vs   bow vertical thruster [volts]' with steps, \
      'mission.output.telemetry'       using 2:26 title 't vs stern vertical thruster [volts]' with steps, \
      'mission.output.telemetry'       using 2:27 title 't vs   bow  lateral thruster [volts]' with steps, \
      'mission.output.telemetry'       using 2:28 title 't vs stern  lateral thruster [volts]' with steps

###########################################################################

pause 0 'gnuplot auv_plot.gnu complete.'
```

IV. UNDERWATER VIRTUAL WORLD GRAPHICS

A. Introduction

The Open Inventor 2.0 applications program interface (API) is used for generating 3D computer graphics images representing the virtual world (Wernecke 94a, 94b). Open Inventor was chosen for the virtual world graphics engine due to its being built upon the Open GL graphics language, expected portability to multiple hardware architectures and operating systems, SGI-standard scene description language, object-oriented design, broad flexibility and strong support. The Open Inventor scene description language (*.iv*) is likely to become a compatible standard with numerous graphics engines and the nascent Virtual Reality Modeling Language (*.vrml*), a research extension to Hypertext Markup Language (*.html*).

Embedded in the graphics viewer code is a Distributed Interactive Simulation (DIS) 2.0.3 multicast network API which permits *viewer* to receive robot update information nearly anywhere the Internet via the Multicast Backbone (MBone). After the configuration file *.sd.tcl* is installed, the *viewer* program can be launched automatically via the MBone session directory (*sd*) advertisement application, enabling simple remote *viewer* execution and connection to an already-running virtual world. Future work includes adding a WWW-compatible scene and object loader, and the capability to receive URLs for retrieval, either via DIS updates or from objects encountered by the robot that are already in the scene graph.

B. *viewer.C* Object-Oriented Real-Time 3D Computer Graphics Viewing Program using Open Inventor 2.0

```
//////////////////////////////////////////////////////////////////////////////////
/*
 Program:          viewer.C

 Description:      Open Inventor viewer for
                   NPS AUV Underwater Virtual World

 Author:           Don Brutzman

 Revised:          28 October 94

 System:           Irix 5.2

 Compiler:         ANSI C++ / Open Inventor API

 Compilation:      irix> make viewer

 References:   (1) IEEE Protocols for Distributed Interactive Simulation (DIS)
                   Applications version 2.0, Institute for Simulation and
                   Training, Universit of Central Florida, Orlando Florida,
                   28 May 1993.

               (2) Macedonia, Michael, Zeswitz, Steven, and Locke, John,
                   Distributed Interactive Simulation (DIS) multicast
                   version 2.0.3, Naval Postgraduate School, February 94.

               (3) Zeswitz, Steven, "NPSNET: Integration of Distributed
                   Interactive Simulation (DIS) Protocol for Communication
                   Architecture and Information Interchange." master's thesis,
                   Naval Postgraduate School, Monterey California, 28 May 1993.

               (4) Wernecke, Josie and the Open Inventor Architecture Group,
                   _The Inventor Mentor_, Addison-Wesley, Reading Massachusetts,
                   1994.

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 Notes:            Underwater virtual world is in feet, standard DIS PDU is meters

                   Inventor 1.0.1 and DIS libraries were NOT compatible due to
                   programs hangups which are triggered by mallocs in the
                   DIS libraries conflicting with the Inventor window.
                   Removing DIS libraries and just putting malloc's in the
                   Inventor screen redraw callback function caused identical
                   program hangups.  No fix was found.  Upgrading to OpenInventor
                   under Irix 5.2 eliminated this problem completely.

 Future work:      Automated camera control

                   .iv file load via WWW URLs

                   optimization

*/
//////////////////////////////////////////////////////////////////////////////////

#include "../dynamics/AUVglobals.H"
#include <iostream.h>
```

```
#include <iomanip.h>  // must follow iostream.h
#include <string.h>
#include <math.h>
#include <time.h>
#include <getopt.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/types.h>

////////////////////////////////////////////////////////////////////////////////

// DIS includes.  See Makefile for other DIS #include files; they must match.

#include "../DIS.mcast/h/disdefs.h"
#include "../DIS.mcast/h/appearance.h"

// Old DIS include statements follow, disregard...

// #include "/n/dude/work/brutzman/DIS.mcast/h/disdefs.h"
// #include "/n/dude/work/brutzman/DIS.mcast/h/appearance.h"

// John Locke's broadcast version
// #include "/n/gravy1/work/simnet/DIS-2.0/h/disdefs.h"
// #include "/n/gravy1/work/simnet/DIS-2.0/h/appearance.h"

// Macedonia/Zeswitz versions
// #include "/n/elsie/work3/macedoni/net/mcast/network/h/disdefs.h"
// #include "/n/elsie/work3/macedoni/net/mcast/network/h/appearance.h"

// #include "/n/elsie/work3/macedoni/net/mcast/network.round/h/disdefs.h"
// #include "/n/elsie/work3/macedoni/net/mcast/network.round/h/appearance.h"

extern "C" { printPDU (char *); }; // function prototype provided for
                                    // compatibility, missing from DIS library

////////////////////////////////////////////////////////////////////////////////

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/CascadeBG.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/Separator.h>
#include <Xm/SeparatoG.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

#include <Inventor/So.h>
#include <Inventor/SoDB.h>
#include <Inventor/Xt/SoXt.h>
#include <Inventor/Xt/SoXtPrintDialog.h> // see SoOffscreenRender
#include <Inventor/Xt/SoXtRenderArea.h>
#include <Inventor/Xt/viewers/SoXtExaminerViewer.h>

#include <Inventor/sensors/SoTimerSensor.h>

#include <Inventor/actions/SoCallbackAction.h>
#include <Inventor/actions/SoWriteAction.h>

#include <Inventor/engines/SoCalculator.h>
#include <Inventor/engines/SoElapsedTime.h>
#include <Inventor/engines/SoTimeCounter.h>

#include <Inventor/nodes/SoComplexity.h>
```

```
#include <Inventor/nodes/SoCone.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoCylinder.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoFaceSet.h>
#include <Inventor/nodes/SoGroup.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoNurbsSurface.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoPickStyle.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoScale.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoSwitch.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoTransformSeparator.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoTriangleStripSet.h> // order matters here
#include <Inventor/nodes/SoUnits.h>

// leftovers from conversion 1.0.1 => OpenInventor

// #include <Inventor/Xt/SoXtColorEditor.h>  // not found!

// #include <Inventor/nodes/SoBase.h>
// #include <Inventor/nodes/SoCallbackList.h>
// #include <Inventor/nodes/SoInteraction.h>
// #include <Inventor/Xt/SoXtFlyViewer.h>
// #include <Inventor/Xt/SoXtPlaneViewer.h>
// #include <Inventor/Xt/SoXtWalkViewer.h>

///////////////////////////////////////////////////////////////////////////////
// function prototypes

static int  DIS_net_open   ();

static void DIS_net_close  ();

static void DIS_Redraw_Callback ( void     * unused_data,
                                  SoSensor * unused_calling_sensor );


///////////////////////////////////////////////////////////////////////////////

#define  PI    3.1415926535897932

#define  METERS_PER_FT  0.3048
#define  FT_PER_METERS  3.2808

// utility function prototypes

double   sign       (double x);

double   degrees    (double x); // radians input

double   radians    (double x); // degrees input

double   arcclamp   (double x);

double   dnormalize (double angle_radians);  // returns 0..2PI

int      inormalize (double angle_radians)   // returns 0..359
                    {return (int) (degrees (angle_radians) + 0.5) % 360;}
```

```
SoSeparator * readFile(const char *filename); // Inventor Mentor p. 284

void     initialize_globals ();

void     parse_command_line_flags (int argc, char ** argv);

//////////////////////////////////////////////////////////////////////////////////
// global data (referenced in callback routines, thus defined here)

SoSeparator          * root;
SoTransform          * AUV_position_node;

// initializers needed to avoid startup segmentation fault

SoRotationXYZ        * rotate_AUV_z = new SoRotationXYZ;
SoRotationXYZ        * rotate_AUV_y = new SoRotationXYZ;
SoRotationXYZ        * rotate_AUV_x = new SoRotationXYZ;

SoRotationXYZ        * rotate_AUV_bow_rudders   = new SoRotationXYZ;
SoRotationXYZ        * rotate_AUV_stern_rudders = new SoRotationXYZ;
SoRotationXYZ        * rotate_AUV_bow_planes    = new SoRotationXYZ;
SoRotationXYZ        * rotate_AUV_stern_planes  = new SoRotationXYZ;

SoCone               * thrusterWakeFV           = new SoCone;
SoCone               * thrusterWakeAV           = new SoCone;
SoCone               * thrusterWakeFH           = new SoCone;
SoCone               * thrusterWakeAH           = new SoCone;
SoCone               * conePropellerWakeStbd    = new SoCone;
SoCone               * conePropellerWakePort    = new SoCone;

SoSwitch             * whichWakeFV              = new SoSwitch;
SoSwitch             * whichWakeAV              = new SoSwitch;
SoSwitch             * whichWakeFH              = new SoSwitch;
SoSwitch             * whichWakeAH              = new SoSwitch;

SoTransform          * topsideBow               = new SoTransform;
SoTransform          * topsideStern             = new SoTransform;
SoTransform          * bottomsideBow            = new SoTransform;
SoTransform          * bottomsideStern          = new SoTransform;
SoTransform          * leftsideBow              = new SoTransform;
SoTransform          * leftsideStern            = new SoTransform;
SoTransform          * rightsideBow             = new SoTransform;
SoTransform          * rightsideStern           = new SoTransform;

SoTransform          * xfConeSonarST1000        = new SoTransform;
SoCone               *   coneSonarST1000        = new SoCone;

SoMaterial * silver    = new SoMaterial;
SoMaterial * gold      = new SoMaterial;
SoMaterial * brass     = new SoMaterial;
SoMaterial * chrome    = new SoMaterial;
SoMaterial * npsblue   = new SoMaterial;
SoMaterial * seagreen  = new SoMaterial;
SoMaterial * darkgreen = new SoMaterial;
SoMaterial * sonar_red = new SoMaterial;

#define  CAMERA_FREE      0
#define  CAMERA_TO_AUV    1
#define  CAMERA_FROM_AUV  2

static int whichCamera            = CAMERA_FREE;  // default camera

static int PRINTDIALOG            = TRUE;

static int BACKGROUNDCOLORDIALOG  = FALSE;

static int AUTOCAMERACONTROL      = FALSE;
```

```
static int TEXTURE                          = FALSE;

SoPerspectiveCamera * PerspectiveCameraFree     = new SoPerspectiveCamera;
SoPerspectiveCamera * PerspectiveCameraToAUV    = new SoPerspectiveCamera;
SoPerspectiveCamera * PerspectiveCameraFromAUV  = new SoPerspectiveCamera;

// SbVec3f cameraToAUVPosition;

SbVec3f     behindAUVPosition;
SbVec3f    aheadOfAUVPosition;
SbVec3f      priorAUVPosition;
SbVec3f    currentAUVPosition;

SbVec3f   priorCameraPosition;
SbVec3f currentCameraPosition;
SbVec3f   priorCameraOffset;
SbVec3f currentCameraOffset;

SbVec3f orientationRotationAxis;
float   orientationRotationAngle;

SbVec3f standardCameraOffset        = SbVec3f ( - 20.0, 0.0, 2.0);

float   standardCameraFocalDistance = 20.0;


SoSeparator * command_line_node;

static clock_t                 time_stamp_of_current_PDU,
                               time_of_PDU_receipt,
                               current_clock,
                               delta_clock;
static double                  delta_time;

static int                     PDU_overdue;

static int                     DIS_port_open;

       char                    port  [ 6];
       char                    group [30];

static SoUnits               * unitsfeet;

static struct in_addr          inaddr;

static struct hostent        * hp;

////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////
// all NPS AUV dimensions here in inches

#define HULLBODYLENGTH   54.00
#define HULLBODYWIDTH    16.50
#define HULLBODYHEIGHT   10.0

#define SEAM            -27.00   // - HULLBODYLENGTH / 2.0
#define STERN           -43.50   // - HULLBODYLENGTH / 2.0 - 16.5
#define LEFT              8.25   //   HULLBODYWIDTH  / 2.0
#define RIGHT            -8.25   // - HULLBODYWIDTH  / 2.0
#define TOP               5.00   //   HULLBODYHEIGHT / 2.0
#define BOTTOM           -5.00   // - HULLBODYHEIGHT / 2.0

#define DOMECONTROLPT     25.00   // nosecone NURBS surface cooordinates
#define DOMECONTROLPTHALF 20.00   //    offsets to help define shape
```

```
#define TOPHALF                 4.75
#define BOTTOMHALF             -4.75
#define LEFT_HALF               8.00
#define RIGHTHALF              -8.00
#define CENTER                  0.00

#define FINLENGTH               6.00   // fins need to be tapered
#define FINWIDTH                0.75   // fin thickness 1" at bottom, 0.5" at top
#define FINHEIGHT               6.75

#define FINOFFSETFORWARD  24.0
#define FINOFFSETAFT     -33.0
#define FINOFFSETLEFT     12.10   // (HULLBODYWIDTH  / 2) + (FINHEIGHT / 2) + .5"
#define FINOFFSETRIGHT   -12.10
#define FINOFFSETUP        8.875  // (HULLBODYHEIGHT / 2) + (FINHEIGHT / 2) + .5"
#define FINOFFSETDOWN     -8.875

#define THRUSTERID              3.0
#define THRUSTEROD              3.5
#define THRUSTERFORWARDV  13.0    // forward SEAM - 14
#define THRUSTERAFTV     - 21.0    // SEAM + 6
#define THRUSTERFORWARDH  19.0    // forward SEAM -  8
#define THRUSTERAFTH     - 26.0    // SEAM + 1

#define SHAFTOFFSETLEFT    3.75
#define SHAFTOFFSETRIGHT  -3.75

// reverify these dimensions after rebuild
#define CARDCAGELEFT        4.00
#define CARDCAGERIGHT      -4.00
#define CARDCAGELENGTH     12.00
#define CARDCAGEWIDTH       7.00
#define CARDCAGEHEIGHT      8.00

// profiling sonar, 1 degree conical
// watch out feet -> inches in code

#define AUV_ST1000_x_offset  34.5   // forward SEAM + 7.5"
#define AUV_ST1000_y_offset  -2.00  //
#define AUV_ST1000_z_offset   4.00  // bottom forward

// scanning sonar sector,  1 degree wide by 24 degree vertical
#define AUV_ST725_x_offset   31.5   // forward SEAM + 4.5"
#define AUV_ST725_y_offset   -2.00  //
#define AUV_ST725_z_offset   -4.00  // top aft

////////////////////////////////////////////////////////////////////////////////
SoSeparator * makeAUV ()
{
   rotate_AUV_z->angle.setValue (  AUV_psi);
   rotate_AUV_z-> axis.setValue (SoRotationXYZ::Z);

   rotate_AUV_y->angle.setValue (- AUV_theta);
   rotate_AUV_y-> axis.setValue (SoRotationXYZ::Y);

   rotate_AUV_x->angle.setValue (  AUV_phi);
   rotate_AUV_x-> axis.setValue (SoRotationXYZ::X);

   SoDrawStyle *wires;
   wires = new SoDrawStyle;
   wires->style = SoDrawStyle::LINES;

////////////////////////////////////////////////////////////////////////////////
//  NPS AUV hull body center is at [0.0  0.0  0.0],
//  volumetric center of buoyancy

   // fin transformations forward
```

```
SoTransform *xf1 = new SoTransform;
xf1->translation.setValue( FINOFFSETFORWARD, 0.0, FINOFFSETUP);
SoTransform *xf2 = new SoTransform;
xf2->translation.setValue( 0.0, 0.0, FINOFFSETDOWN - FINOFFSETUP);

SoTransform *xf3 = new SoTransform;
xf3->translation.setValue( FINOFFSETFORWARD, FINOFFSETLEFT, 0.0);
SoTransform *xf4 = new SoTransform;
xf4->translation.setValue( 0.0, FINOFFSETRIGHT - FINOFFSETLEFT, 0.0);

// fin transformations aft

SoTransform *xf5 = new SoTransform;
xf5->translation.setValue( FINOFFSETAFT, 0.0, FINOFFSETUP);
SoTransform *xf6 = new SoTransform;
xf6->translation.setValue( 0.0 , 0.0, FINOFFSETDOWN - FINOFFSETUP);

SoTransform *xf7 = new SoTransform;
xf7->translation.setValue( FINOFFSETAFT, FINOFFSETLEFT, 0.0);
SoTransform *xf8 = new SoTransform;
xf8->translation.setValue( 0.0, FINOFFSETRIGHT - FINOFFSETLEFT, 0.0);

//  90 degree increment rotations - get #define'd PI values

SoRotationXYZ *ro90x = new SoRotationXYZ;
ro90x->angle.setValue (3.141592653 / 2.0);
ro90x-> axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro90y = new SoRotationXYZ;
ro90y->angle.setValue (3.141592653 / 2.0);
ro90y-> axis.setValue (SoRotationXYZ::Y);

SoRotationXYZ *ro90z = new SoRotationXYZ;
ro90z->angle.setValue (3.141592653 / 2.0);
ro90z-> axis.setValue (SoRotationXYZ::Z);

SoRotationXYZ *ro180x = new SoRotationXYZ;
ro180x->angle.setValue (3.141592653);
ro180x-> axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro180y = new SoRotationXYZ;
ro180y->angle.setValue (3.141592653);
ro180y-> axis.setValue (SoRotationXYZ::Y);

SoRotationXYZ *ro180z = new SoRotationXYZ;
ro180z->angle.setValue (3.141592653);
ro180z-> axis.setValue (SoRotationXYZ::Z);

SoRotationXYZ *ro270x = new SoRotationXYZ;
ro270x->angle.setValue (- 3.141592653 / 2.0);
ro270x-> axis.setValue (SoRotationXYZ::X);

SoRotationXYZ *ro270y = new SoRotationXYZ;
ro270y->angle.setValue (- 3.141592653 / 2.0);
ro270y-> axis.setValue (SoRotationXYZ::Y);

SoRotationXYZ *ro270z = new SoRotationXYZ;
ro270z->angle.setValue (- 3.141592653 / 2.0);
ro270z-> axis.setValue (SoRotationXYZ::Z);

// construct NPS AUV hull body
SoCube *hull = new SoCube;
hull->width  = HULLBODYLENGTH;
hull->height = HULLBODYWIDTH;
hull->depth  = HULLBODYHEIGHT;

// construct a control fin
```

```
SoCube *fin = new SoCube;
fin->width  = FINLENGTH;
fin->height = FINWIDTH;
fin->depth  = FINHEIGHT;

// construct fin pairs

rotate_AUV_bow_rudders = new SoRotationXYZ;
rotate_AUV_bow_rudders->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_bow_rudders->angle.setValue ( 0.3 );

rotate_AUV_stern_rudders = new SoRotationXYZ;
rotate_AUV_stern_rudders->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_stern_rudders->angle.setValue ( 0.6 );

rotate_AUV_bow_planes = new SoRotationXYZ;
rotate_AUV_bow_planes->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_bow_planes->angle.setValue ( 0.9 );

rotate_AUV_stern_planes = new SoRotationXYZ;
rotate_AUV_stern_planes->axis.setValue (SoRotationXYZ::Z);
rotate_AUV_stern_planes->angle.setValue ( 1.2 );

// construct forward vertical fins (bow rudders)
SoSeparator *fvfins = new SoSeparator;
fvfins->addChild( xf1 );
fvfins->addChild( rotate_AUV_bow_rudders );
fvfins->addChild( fin );
fvfins->addChild( xf2 );
fvfins->addChild( ro180x );     // net rotation  180
fvfins->addChild( fin );

// construct aft vertical fins (stern rudders)
SoSeparator *avfins = new SoSeparator;
avfins->addChild( xf5 );
avfins->addChild( rotate_AUV_stern_rudders );
avfins->addChild( fin );
avfins->addChild( xf6 );
avfins->addChild( ro180x );     // net rotation  180
avfins->addChild( fin );

// construct forward horizontal fins (bow planes)
SoSeparator *fhfins = new SoSeparator;
fhfins->addChild( xf3 );
fhfins->addChild( ro90x );      // net rotation   90
fhfins->addChild( rotate_AUV_bow_planes );
fhfins->addChild( fin );
fhfins->addChild( ro270x );
fhfins->addChild( xf4 );
fhfins->addChild( ro270x );     // net rotation  270 (in case of fin asymmetry)

fhfins->addChild( fin );

// construct aft horizontal fins (stern planes)
SoSeparator *ahfins = new SoSeparator;
ahfins->addChild( xf7 );
ahfins->addChild( ro90x );     // net rotation  90
ahfins->addChild( rotate_AUV_stern_planes );
ahfins->addChild( fin );
ahfins->addChild( ro270x );
ahfins->addChild( xf8 );
ahfins->addChild( ro270x );    // net rotation 270 (in case of fin asymmetry)
ahfins->addChild( fin );

// construct cylinders to represent the thrusters
SoTransform *xf13 = new SoTransform;
xf13->translation.setValue(THRUSTERFORWARDV, 0.0, 0.0);
```

```
SoTransform *xf14 = new SoTransform;
xf14->translation.setValue(THRUSTERAFTV, 0.0, 0.0);
SoTransform *xf15 = new SoTransform;
xf15->translation.setValue(THRUSTERFORWARDH, 0.0, 0.0);
SoTransform *xf16 = new SoTransform;
xf16->translation.setValue(THRUSTERAFTH, 0.0, 0.0);

SoCylinder * tubeV = new SoCylinder;
tubeV->radius = THRUSTERID;
tubeV->height = HULLBODYHEIGHT + 0.2;
SoCylinder * tubeH = new SoCylinder;
tubeH->radius = THRUSTERID;
tubeH->height = HULLBODYWIDTH  + 0.2;

SoComplexity * wakeComplexity = new SoComplexity;
wakeComplexity->value = 0.2;

// still inches
topsideBow->      translation.setValue( 0,     TOPHALF + AUV_bow_vertical, 0);
bottomsideBow->   translation.setValue( 0, BOTTOMHALF + AUV_bow_vertical, 0);
bottomsideBow->      rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0 ), M_PI );

topsideStern->    translation.setValue( 0,     TOPHALF + AUV_stern_vertical,0);
bottomsideStern->translation.setValue( 0, BOTTOMHALF + AUV_stern_vertical,0);
bottomsideStern->    rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0), M_PI );

leftsideBow->     translation.setValue( 0, LEFT_HALF + AUV_bow_lateral, 0);
leftsideBow->        rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0 ), M_PI );
rightsideBow->    translation.setValue( 0, RIGHTHALF + AUV_bow_lateral, 0);

leftsideStern->   translation.setValue( 0, LEFT_HALF + AUV_stern_lateral, 0);
leftsideStern->      rotation.setValue(SbVec3f ( 0.0, 1.0, 0.0  ), M_PI );
rightsideStern-> translation.setValue( 0, RIGHTHALF + AUV_stern_lateral, 0);

thrusterWakeFV = new SoCone; // global for callbacks
thrusterWakeFV->height       = AUV_bow_vertical * 2.0;
thrusterWakeFV->bottomRadius = AUV_bow_vertical / 4.0;
thrusterWakeFV->parts        = SoCone::SIDES;

whichWakeFV = new SoSwitch;
whichWakeFV->addChild (topsideBow);
whichWakeFV->addChild (bottomsideBow);
whichWakeFV->whichChild = 0; // default topsideBow

SoSeparator * thrusterFV = new SoSeparator;
thrusterFV->addChild( xf13 );
thrusterFV->addChild( ro90x );
thrusterFV->addChild( tubeV );
thrusterFV->addChild( wires );
thrusterFV->addChild( wakeComplexity );
thrusterFV->addChild( seagreen );
thrusterFV->addChild( whichWakeFV );
thrusterFV->addChild( thrusterWakeFV );

thrusterWakeAV = new SoCone; // global for callbacks
thrusterWakeAV->height       = AUV_stern_vertical * 2.0;
thrusterWakeAV->bottomRadius = AUV_stern_vertical / 4.0;
thrusterWakeAV->parts        = SoCone::SIDES;

whichWakeAV = new SoSwitch;
whichWakeAV->addChild (topsideStern);
whichWakeAV->addChild (bottomsideStern);
whichWakeAV->whichChild = 0; // default topsideStern

SoSeparator * thrusterAV = new SoSeparator;
thrusterAV->addChild( xf14 );
thrusterAV->addChild( ro90x );
```

```
thrusterAV->addChild( tubeV );
thrusterAV->addChild( wires );
thrusterAV->addChild( wakeComplexity );
thrusterAV->addChild( seagreen );
thrusterAV->addChild( whichWakeAV );
thrusterAV->addChild( thrusterWakeAV );

thrusterWakeFH = new SoCone; // global for callbacks
thrusterWakeFH->height       = AUV_bow_lateral * 2.0;
thrusterWakeFH->bottomRadius = AUV_bow_lateral / 4.0;
thrusterWakeFH->parts        = SoCone::SIDES;

whichWakeFH = new SoSwitch;
whichWakeFH->addChild (leftsideBow);
whichWakeFH->addChild (rightsideBow);
whichWakeFH->whichChild = 0; // default leftsideBow

SoSeparator * thrusterFH = new SoSeparator;
thrusterFH->addChild( xf15 );
thrusterFH->addChild( ro90y );
thrusterFH->addChild( tubeH );
thrusterFH->addChild( ro180z );
thrusterFH->addChild( wires );
thrusterFH->addChild( wakeComplexity );
thrusterFH->addChild( seagreen );
thrusterFH->addChild( whichWakeFH );
thrusterFH->addChild( thrusterWakeFH );

thrusterWakeAH = new SoCone; // global for callbacks
thrusterWakeAH->height       = AUV_stern_lateral * 2.0;
thrusterWakeAH->bottomRadius = AUV_stern_lateral / 4.0;
thrusterWakeAH->parts        = SoCone::SIDES;

whichWakeAH = new SoSwitch;
whichWakeAH->addChild (leftsideStern);
whichWakeAH->addChild (rightsideStern);
whichWakeAH->whichChild = 0; // default leftsideStern

SoSeparator * thrusterAH = new SoSeparator;
thrusterAH->addChild( xf16 );
thrusterAH->addChild( ro90y );
thrusterAH->addChild( tubeH );
thrusterAH->addChild( ro180z );
thrusterAH->addChild( wires );
thrusterAH->addChild( wakeComplexity );
thrusterAH->addChild( seagreen );
thrusterAH->addChild( whichWakeAH );
thrusterAH->addChild( thrusterWakeAH );

// construct internal CARDCAGEs left and right

SoCube *cardcageleftbox = new SoCube;
cardcageleftbox->width  = CARDCAGELENGTH;
cardcageleftbox->height = CARDCAGEWIDTH;
cardcageleftbox->depth  = CARDCAGEHEIGHT;

SoTransform *xfcardcageleft = new SoTransform;
xfcardcageleft->translation.setValue( 0.0, CARDCAGELEFT, 0.0 );

SoSeparator *cardcageleft = new SoSeparator;
cardcageleft->addChild( xfcardcageleft );
cardcageleft->addChild( cardcageleftbox );

SoCube *cardcagerightbox = new SoCube;
cardcagerightbox->width  = CARDCAGELENGTH;
cardcagerightbox->height = CARDCAGEWIDTH;
cardcagerightbox->depth  = CARDCAGEHEIGHT;
```

```
    SoTransform *xfcardcageright = new SoTransform;
    xfcardcageright->translation.setValue( 0.0, CARDCAGERIGHT, 0.0 );

    SoSeparator *cardcageright = new SoSeparator;
    cardcageright->addChild( xfcardcageright );
    cardcageright->addChild( cardcagerightbox );

///////////////////////////////////////////////////////////////////////////////////
// construct main body out of parts
    SoGroup *body = new SoGroup;

    body->addChild( gold );
    body->addChild( fvfins );
    body->addChild( avfins );
    body->addChild( fhfins );
    body->addChild( ahfins );

    body->addChild( npsblue );
    body->addChild( hull );

    body->addChild( silver );
    body->addChild( thrusterFV );
    body->addChild( thrusterAV );
    body->addChild( thrusterFH );
    body->addChild( thrusterAH );
    body->addChild( cardcageleft );
    body->addChild( cardcageright );

///////////////////////////////////////////////////////////////////////////////////
// construct nosecone using a NURBS surface

    static float pts [25][3] =
    {
        { - SEAM,                          LEFT,       TOP        },
        { - SEAM,                          LEFT_HALF,  TOP        },
        { - SEAM,                          CENTER,     TOP        },
        { - SEAM,                          RIGHTHALF,  TOP        },
        { - SEAM,                          RIGHT,      TOP        },

        { - SEAM,                          LEFT,       TOPHALF    },
        { - SEAM + DOMECONTROLPTHALF,      LEFT_HALF,  TOPHALF    },
        { - SEAM + DOMECONTROLPTHALF,      CENTER,     TOPHALF    },
        { - SEAM + DOMECONTROLPTHALF,      RIGHTHALF,  TOPHALF    },
        { - SEAM,                          RIGHT,      TOPHALF    },

        { - SEAM,                          LEFT,       CENTER     },
        { - SEAM + DOMECONTROLPTHALF,      LEFT_HALF,  CENTER     },
        { - SEAM + DOMECONTROLPT,          CENTER,     CENTER     },
        { - SEAM + DOMECONTROLPTHALF,      RIGHTHALF,  CENTER     },
        { - SEAM,                          RIGHT,      CENTER     },

        { - SEAM,                          LEFT,       BOTTOMHALF },
        { - SEAM + DOMECONTROLPTHALF,      LEFT_HALF,  BOTTOMHALF },
        { - SEAM + DOMECONTROLPTHALF,      CENTER,     BOTTOMHALF },
        { - SEAM + DOMECONTROLPTHALF,      RIGHTHALF,  BOTTOMHALF },
        { - SEAM,                          RIGHT,      BOTTOMHALF },

        { - SEAM,                          LEFT,       BOTTOM     },
        { - SEAM,                          LEFT_HALF,  BOTTOM     },
        { - SEAM,                          CENTER,     BOTTOM     },
        { - SEAM,                          RIGHTHALF,  BOTTOM     },
        { - SEAM,                          RIGHT,      BOTTOM     },
    };

    static float knots [10] =
```

```
            { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 };

SoComplexity *noseconeComplexity = new SoComplexity;
noseconeComplexity->value = 0.7;

SoCoordinate3 *controlPts = new SoCoordinate3;
controlPts->point.setValues ( 0, 25, pts );

SoNurbsSurface *sonardome = new SoNurbsSurface;
sonardome->numUControlPoints.setValue ( 5 );
sonardome->numVControlPoints.setValue ( 5 );
sonardome->uKnotVector.setValues ( 0, 10, knots );
sonardome->vKnotVector.setValues ( 0, 10, knots );

SoSeparator *nosesection = new SoSeparator;
nosesection->addChild( npsblue );
nosesection->addChild( noseconeComplexity );
nosesection->addChild( controlPts );
nosesection->addChild( sonardome );

coneSonarST1000 = new SoCone; // global for callbacks
coneSonarST1000->height       = fabs (AUV_ST1000_range);
coneSonarST1000->bottomRadius = fabs (AUV_ST1000_range) / 60.0; // 1 degree
coneSonarST1000->parts        = SoCone::SIDES;

// drawn from center
xfConeSonarST1000->translation.setValue( 0.0,
                                   // - (HULLBODYLENGTH  / 2.0)
                                      - (AUV_ST1000_range/ 2.0), 4.0/12.0);

SoSeparator * sepSonar = new SoSeparator;
sepSonar->addChild( ro90z );
sepSonar->addChild( wires );
sepSonar->addChild( wakeComplexity );
sepSonar->addChild( sonar_red );
sepSonar->addChild( xfConeSonarST1000 );
sepSonar->addChild( coneSonarST1000 );

//////////////////////////////////////////////////////////////////////////////////////
// Define tail section

// ensure polygons are defined in clockwise fashion!

// Two triangles using SoTriangleStripSet:
static long numbertrianglevertices [2] = {3, 3};

static float afttrianglevertices [6][3] =
{
 {STERN, LEFT,  0.0}, {SEAM, LEFT,  TOP},    {SEAM,  LEFT,  BOTTOM},
 {STERN, RIGHT, 0.0}, {SEAM, RIGHT, BOTTOM}, {SEAM,  RIGHT, TOP},
};

// Define coordinates for triangular vertices & SoTriangleStripSet
SoCoordinate3 *tailcoord1 = new SoCoordinate3;
tailcoord1->point.setValues (0, 6, afttrianglevertices);

SoTriangleStripSet *tailtriangleset = new SoTriangleStripSet;
tailtriangleset->numVertices.setValues
                      (0, 2, numbertrianglevertices);

// Two rectangles using FaceSet:
static long numberquadvertices [2] = {4, 4};  // ref. p. 5-6

static float aftquadvertices [8][3] =
{
 {STERN, LEFT,  0.0},  {STERN, RIGHT, 0.0},  {SEAM,  RIGHT, TOP},
 {SEAM,  LEFT, TOP},
```

```
 {STERN, RIGHT, 0.0},  {STERN, LEFT,  0.0},  {SEAM,  LEFT, BOTTOM},
 {SEAM,  RIGHT, BOTTOM},
};

// Define coordinates for quad vertices & SoFaceSet
SoCoordinate3 *tailcoord2 = new SoCoordinate3;
tailcoord2->point.setValues (0, 8, aftquadvertices);

SoFaceSet *tailquadset = new SoFaceSet;
tailquadset->numVertices.setValues (0, 2, numberquadvertices);

// Two cylinders currently represent the propellors - improve this!
// a much fancier individual prop model is possible here; also add complexity
SoCylinder *prop = new SoCylinder;
prop->radius = 2.00;
prop->height = 1.00;

// Two cylinders to represent the shafts
SoCylinder *shaft = new SoCylinder;
shaft->radius = 0.50;
shaft->height = 4.00;

SoTransform *xf18 = new SoTransform;  // shafts relative to props
                                      // note:  rotated 90z
xf18->translation.setValue(0.0, -2.0, 0.0);

SoTransform *xf11 = new SoTransform;  // left prop
xf11->translation.setValue(STERN - 2.0, SHAFTOFFSETLEFT,  0.0);

// compose shaft with prop
SoSeparator *leftprop = new SoSeparator;
leftprop->addChild( xf11 );
leftprop->addChild( ro90z );
leftprop->addChild( prop );
leftprop->addChild( xf18 );
leftprop->addChild( shaft );

SoTransform *xfPropellerWakePort = new SoTransform;
xfPropellerWakePort->translation.setValue( 0.0, 15.0, 0.0 );
SoSeparator * separatorPropellerWakePort = new SoSeparator;
separatorPropellerWakePort->addChild( xfPropellerWakePort );
separatorPropellerWakePort->addChild( ro180x );
separatorPropellerWakePort->addChild( wires );
separatorPropellerWakePort->addChild( wakeComplexity );
separatorPropellerWakePort->addChild( seagreen );
conePropellerWakePort = new SoCone; // global for callbacks
conePropellerWakePort->height       =        AUV_port_rpm  / 700.0 * 24.0;
conePropellerWakePort->bottomRadius = fabs (AUV_port_rpm) / 700.0 *  6.0;
conePropellerWakePort->parts        = SoCone::SIDES;
separatorPropellerWakePort->addChild( conePropellerWakePort );
leftprop->addChild( separatorPropellerWakePort );

SoTransform *xf12 = new SoTransform;  // right props
xf12->translation.setValue(STERN - 2.0, SHAFTOFFSETRIGHT,  0.0);

SoSeparator *rightprop = new SoSeparator;
rightprop->addChild( xf12 );
rightprop->addChild( ro90z );
rightprop->addChild( prop );
rightprop->addChild( xf18 );
rightprop->addChild( shaft );

SoTransform *xfPropellerWakeStbd = new SoTransform;
xfPropellerWakeStbd->translation.setValue( 0.0, 15.0, 0.0 );
SoSeparator * separatorPropellerWakeStbd = new SoSeparator;
separatorPropellerWakeStbd->addChild( xfPropellerWakeStbd );
separatorPropellerWakeStbd->addChild( ro180x );
```

```
    separatorPropellerWakeStbd->addChild( wires );
    separatorPropellerWakeStbd->addChild( wakeComplexity );
    separatorPropellerWakeStbd->addChild( seagreen );
    conePropellerWakeStbd = new SoCone; // global for callbacks
    conePropellerWakeStbd->height       =        AUV_port_rpm   / 700.0 * 24.0;
    conePropellerWakeStbd->bottomRadius = fabs (AUV_port_rpm) / 700.0 *  6.0;
    conePropellerWakeStbd->parts        = SoCone::SIDES;
    separatorPropellerWakeStbd->addChild( conePropellerWakeStbd );
    rightprop->addChild( separatorPropellerWakeStbd );

    // construct stern box support beneath aft vertical fins
    SoCube *sternfinsupportcube = new SoCube;
    sternfinsupportcube->width  = FINLENGTH + 1.5;
    sternfinsupportcube->height = 5.0;
    sternfinsupportcube->depth  = HULLBODYHEIGHT - 0.5;

    SoTransform *xf17 = new SoTransform;
    xf17->translation.setValue(FINOFFSETAFT, 0.0,  0.0);

    SoSeparator *sternfinsupport = new SoSeparator;
    sternfinsupport->addChild( xf17 );
    sternfinsupport->addChild( sternfinsupportcube );

    SoSeparator *tailsection = new SoSeparator;
    tailsection->addChild( npsblue );
    tailsection->addChild( tailcoord1 );
    tailsection->addChild( tailtriangleset );
    tailsection->addChild( tailcoord2 );
    tailsection->addChild( tailquadset );
    tailsection->addChild( sternfinsupport );
    tailsection->addChild( brass );
    tailsection->addChild( leftprop );
    tailsection->addChild( rightprop );

////////////////////////////////////////////////////////////////////////////////////
// robot is just units & body & nosesection & tailsection & extra stuff

    SoSeparator * robot = new SoSeparator;

    robot->addChild( unitsfeet );

    // robot root transform for overall vehicle orientation
    AUV_position_node = new SoTransform;
    AUV_position_node->translation.setValue(0.0, 0.0, 0.0);
    robot->addChild( AUV_position_node );

    rotate_AUV_z = new SoRotationXYZ;
    rotate_AUV_z->angle.setValue ( - AUV_psi);
    rotate_AUV_z-> axis.setValue (SoRotationXYZ::Z);
    robot->addChild( rotate_AUV_z );

    rotate_AUV_y = new SoRotationXYZ;
    rotate_AUV_y->angle.setValue ( - AUV_theta);
    rotate_AUV_y-> axis.setValue (SoRotationXYZ::Y);
    robot->addChild( rotate_AUV_y );

    rotate_AUV_x = new SoRotationXYZ;
    rotate_AUV_x->angle.setValue (   AUV_phi);
    rotate_AUV_x-> axis.setValue (SoRotationXYZ::X);
    robot->addChild( rotate_AUV_x );

    robot->addChild( sepSonar );  // feet

    SoUnits *unitsinches = new SoUnits;
    unitsinches->units.setValue ( SoUnits::INCHES );
    robot->addChild( unitsinches );
```

```
   SoPickStyle * unpickablestylenode;
   SoPickStyle *   pickablestylenode;
   unpickablestylenode = new SoPickStyle;
     pickablestylenode = new SoPickStyle;
   unpickablestylenode->style.setValue ( SoPickStyle::UNPICKABLE );
     pickablestylenode->style.setValue ( SoPickStyle::BOUNDING_BOX );

   // Make subsequent nodes unpickable so that AUV is treated as a whole
   robot->addChild( unpickablestylenode );

   robot->addChild( body );
   robot->addChild( nosesection );
   robot->addChild( tailsection );

   SoTransform *xf19 = new SoTransform;
   xf19->translation.setValue( 0.0, 0.0, - 2.0);
   robot->addChild( xf19 );
   robot->addChild( new SoPointLight );
   cout << "new point light added under robot" << endl;


   return robot;
}
//////////////////////////////////////////////////////////////////////////////

double sign (double x)
{
   if      (x > 0.0) return  1.0;
   else if (x < 0.0) return -1.0;
   else              return  1.0;
}
//--------------------------------------------------------------------------//

double degrees (double x)     // radians input
{
   return x * 180.0 / PI;
}
//--------------------------------------------------------------------------//

double radians (double x)    // degrees input
{
   return x * PI / 180.0;
}
//--------------------------------------------------------------------------//

double arcclamp (double x)
{
   if      (x >  1.0)
   {
      x =  1.0;
      cout << "viewer:  arcclamp reduced " << x << " to 1.0" << endl;
   }
   else if (x < -1.0)
   {
      x = -1.0;
      cout << "viewer:  arcclamp raised " << x << " to -1.0" << endl;
   }
   return x;
}
//--------------------------------------------------------------------------//

double dnormalize (double angle_radians)
```

```
{
   double new_angle = angle_radians;

   while (new_angle > 2*PI) new_angle -= 2*PI;
   while (new_angle < 0.0 ) new_angle += 2*PI;
   return new_angle;
}
/////////////////////////////////////////////////////////////////////////////////

//---------------------------------------------------------------------------//

static int DIS_net_open ()  // Ref: macedonia include files
{
   // Multicast Defaults from
   // /n/elsie/work3/macedoni/net/mcast/network/utils/planes/planes.cc

   //     ttl value does not matter since this viewing program only reads PDUs
   u_char ttl                = 16;        // multicast ttl=16 stays inside NPS

   int    exercise_id        = -1;
   int    coordinate_system = 0;  // 0 = flat world, 1 = round world
   char * utm_file           = "";

   int  result = net_open (port, group, ttl,
                           exercise_id, coordinate_system, utm_file);

// int  result = net_open (port, group, ttl); // old version

   if   result == FALSE)
   {
        cout << "viewer: DIS_net_open () failed" << endl;
        DIS_port_open = FALSE;
   }
   else
   {
     DIS_port_open = TRUE;

       // cout << "viewer: port = "  << port     << ", group = " << group
       //      << ", ttl = "       << ttl      << endl;
       // cout << "  exercise_id = "          << exercise_id
       //      << ", coordinate_system = "    << coordinate_system
       //      << ", utm_file = \"" << utm_file << "\"" << endl;
   }
   return result;
}
//---------------------------------------------------------------------------//

static void DIS_Redraw_Callback ( void     * unused_data,
                                  SoSensor * unused_calling_sensor )
{
   double delta_x     = 0.0;
   double delta_y     = 0.0;
   double delta_z     = 0.0;
   double delta_phi   = 0.0;
   double delta_theta = 0.0;
   double delta_psi   = 0.0;

   int                       number_of_PDUs;

   EntityID                  UUV_DIS_id;

   EntityType                UUV_DIS_type;

   EntityStatePDU          * UUV_DIS_pdu = NULL;

   char                    * local_PDU   = NULL;
```

```
PDUType                          local_PDU_type;

static int rcvd = 0;

char       NPSAUV_Marking [8];
bzero      (NPSAUV_Marking, 8);
strncpy    (NPSAUV_Marking, "NPS AUV", 7); // 4 more free chars remain

if (delta_time <= 5.5)
    cout << "viewer: DIS_Redraw_Callback: PDU loop delta_time = "
         << delta_time << endl;

while (TRUE)  // until break
{
   number_of_PDUs = net_read (& local_PDU, & local_PDU_type); // old version

// number_of_PDUs = net_read (& local_PDU, & local_PDU_type, & inaddr);

   // cout << "viewer: net_read complete, number_of_PDUs available = "
   //      << number_of_PDUs << endl;

   if (number_of_PDUs == -1)
       cout << "viewer: Error on net_read, number_of_PDUs = -1" << endl;

   if (number_of_PDUs <= 0)
   {
       break;  // no more PDUs, done for now
   }
   // rcvd ++;
   // cout << "PDU received, rcvd = " << rcvd << endl;
   // printPDU (local_PDU);

   UUV_DIS_pdu = (EntityStatePDU *) local_PDU;

   // ensure the PDU values are the right types
   if (local_PDU_type != EntityStatePDU_Type)
   {
       cout << "viewer: local_PDU_type != EntityStatePDU_Type, ignored..."
            << endl;

       // don't forget this or get a memory leak!
       // articulated parameters are also freed
       freePDU ((char *) UUV_DIS_pdu);
       cout << "viewer: freePDU ((char *) UUV_DIS_pdu) called for this PDU"
            << endl;

       continue;  // not a break since other PDUs may be waiting
   }
   else if (strncmp ((char *) UUV_DIS_pdu->entity_marking.markings,
                          NPSAUV_Marking, 7) != 0)
   {
       cout << "viewer: non-NPS AUV Entity State PDU encountered, ignored..."
            << endl;
       // printPDU (local_PDU);
       freePDU ((char *) UUV_DIS_pdu);
       // don't forget this or get a memory leak!
       // articulated parameters are also freed
       cout << "viewer: freePDU ((char *) UUV_DIS_pdu) called for this PDU"
            << endl;

       continue;  // not a break since other PDUs may be waiting
   }
   // cout << "PDU OK" << endl;

   // extract parameters of an entity state PDU (most are listed in pdu.h)
   //     this assumes there are no articulated parameters (add later) <<<
```

```
// DIS ID and Type
// UUV_DIS_id   = UUV_DIS_pdu->entity_id;
// UUV_DIS_type = UUV_DIS_pdu->entity_type;

time_stamp_of_current_PDU = UUV_DIS_pdu->entity_state_header.time_stamp;
time_of_PDU_receipt       = clock ();

PDU_overdue = FALSE;

// Posture
AUV_x       =         UUV_DIS_pdu->entity_location.x * FT_PER_METERS;
AUV_y       =         UUV_DIS_pdu->entity_location.y * FT_PER_METERS;
AUV_z       =         UUV_DIS_pdu->entity_location.z * FT_PER_METERS;
AUV_phi     = radians (UUV_DIS_pdu->entity_orientation.phi);
AUV_theta   = radians (UUV_DIS_pdu->entity_orientation.theta);
AUV_psi     = radians (UUV_DIS_pdu->entity_orientation.psi);

cout << "viewer: DIS_net_read posture trace:" << endl;
cout << "[";
cout <<   UUV_DIS_pdu->entity_location.x       << ", ";
cout <<   UUV_DIS_pdu->entity_location.y       << ", ";
cout <<   UUV_DIS_pdu->entity_location.z       << ", ";
cout <<   UUV_DIS_pdu->entity_orientation.phi  << ", ";
cout <<   UUV_DIS_pdu->entity_orientation.theta << ", ";
cout <<   UUV_DIS_pdu->entity_orientation.psi  << "]" << endl;

// Linear and angular velocities in body coordinates/meters by DIS standard
AUV_x_dot              = UUV_DIS_pdu->entity_velocity.x * FT_PER_METERS;
AUV_y_dot              = UUV_DIS_pdu->entity_velocity.y * FT_PER_METERS;
AUV_z_dot              = UUV_DIS_pdu->entity_velocity.z * FT_PER_METERS;

AUV_phi_dot   =radians(UUV_DIS_pdu->dead_reckon_params.angular_velocity[0]);

AUV_theta_dot=radians(UUV_DIS_pdu->dead_reckon_params.angular_velocity[1]);

AUV_psi_dot   =radians(UUV_DIS_pdu->dead_reckon_params.angular_velocity[2]);

//     cout << "viewer: World coordinate velocities:   ";
//     cout << "[";
//     cout <<   UUV_DIS_pdu->entity_velocity.x        << ", ";
//     cout <<   UUV_DIS_pdu->entity_velocity.y        << ", ";
//     cout <<   UUV_DIS_pdu->entity_velocity.z        << ", ";
//     cout <<   UUV_DIS_pdu->dead_reckon_params.angular_velocity [0] << ", ";
//     cout <<   UUV_DIS_pdu->dead_reckon_params.angular_velocity [1] << ", ";
//     cout <<   UUV_DIS_pdu->dead_reckon_params.angular_velocity [2] << "]"
//          << endl;
//     cout << endl;

// Note that even though the accelerations are calculated in the superclass
// UUVBody, use of global state vector lets us construct a class hierarchy
// based on problem structure instead of the communications dependencies.
// This is proposed as a general DIS-compatible vehicle object hierarchy.

// Accelerations are not produced in world coordinates, thus zeroes expected
AUV_u_dot = UUV_DIS_pdu->dead_reckon_params.linear_accel [0] * FT_PER_METERS;
AUV_v_dot = UUV_DIS_pdu->dead_reckon_params.linear_accel [1] * FT_PER_METERS;
AUV_w_dot = UUV_DIS_pdu->dead_reckon_params.linear_accel [2] * FT_PER_METERS;

//     cout << "viewer: World coordinate accelerations: ";
//     cout << "[";
//     cout <<   UUV_DIS_pdu->dead_reckon_params.linear_accel [0] << ", ";
//     cout <<   UUV_DIS_pdu->dead_reckon_params.linear_accel [1] << ", ";
//     cout <<   UUV_DIS_pdu->dead_reckon_params.linear_accel [2] << "]" << endl;

// what we look like
// UUV_DIS_pdu->entity_appearance;
// UUV_DIS_pdu->entity_marking.character_set;
```

```
// UUV_DIS_pdu->entity_marking.markings;

// project our movement
UUV_DIS_pdu->dead_reckon_params.algorithm = DRAlgo_DRM_FPW;

// cout << "viewer: DIS_net_read () successful" << endl;
// cout << flush;

// update overall AUV posture (both position & orientation)
AUV_position_node->translation.setValue ( AUV_x,    - AUV_y, - AUV_z);
rotate_AUV_z->            angle.setValue ( - AUV_psi);
rotate_AUV_y->            angle.setValue ( - AUV_theta);
rotate_AUV_x->            angle.setValue (   AUV_phi);

ArticulatParamsNode * APNptr = UUV_DIS_pdu->articulat_params_head;

AUV_time          =   APNptr->articulat_params.parameter_value [0];
AUV_time         +=   APNptr->articulat_params.parameter_value [1] / 10.0;
AUV_delta_rudder =   APNptr->articulat_params.parameter_value [2];
AUV_delta_planes =   APNptr->articulat_params.parameter_value [3];

// denormalize former shorts to be negative if necessary
if (AUV_delta_rudder >= 128.0) AUV_delta_rudder -= 256.0;
if (AUV_delta_planes >= 128.0) AUV_delta_planes -= 256.0;

AUV_port_rpm        = APNptr->articulat_params.parameter_value [4];
if  (AUV_port_rpm  >= 128.0) AUV_port_rpm = AUV_port_rpm - 256.0;
AUV_port_rpm       *= 10.0;
if  (AUV_port_rpm  >= 0.0)
     AUV_port_rpm  += APNptr->articulat_params.parameter_value [5];
else AUV_port_rpm  -= APNptr->articulat_params.parameter_value [5];

AUV_stbd_rpm        = APNptr->articulat_params.parameter_value [6];
if  (AUV_stbd_rpm >= 128.0) AUV_stbd_rpm = AUV_stbd_rpm - 256.0;
AUV_stbd_rpm       *= 10.0;
if  (AUV_stbd_rpm >= 0.0)
     AUV_stbd_rpm += APNptr->articulat_params.parameter_value [7];
else AUV_stbd_rpm -= APNptr->articulat_params.parameter_value [7];

// cout << "viewer: Articulation parameter 0:" << endl;
   cout << "AUV_time          = " << AUV_time          << endl;
// cout << "AUV_delta_rudder = " << AUV_delta_rudder << endl;
// cout << "AUV_delta_planes = " << AUV_delta_planes << endl;
// cout << "AUV_port_rpm     = " << AUV_port_rpm     << endl;
// cout << "AUV_stbd_rpm     = " << AUV_stbd_rpm    << endl;

APNptr = APNptr->next_articulat_params; // next articulated parameter

// thrusters
AUV_bow_vertical    = APNptr->articulat_params.parameter_value [0];
AUV_stern_vertical = APNptr->articulat_params.parameter_value [1];
AUV_bow_lateral     = APNptr->articulat_params.parameter_value [2];
AUV_stern_lateral   = APNptr->articulat_params.parameter_value [3];

// denormalize former shorts to be negative if necessary
if (AUV_bow_vertical    >= 128.0) AUV_bow_vertical    -= 256.0;
if (AUV_stern_vertical >= 128.0) AUV_stern_vertical -= 256.0;
if (AUV_bow_lateral     >= 128.0) AUV_bow_lateral     -= 256.0;
if (AUV_stern_lateral   >= 128.0) AUV_stern_lateral   -= 256.0;

// convert thruster volts to force by signed squares & normalize adjust
AUV_bow_vertical    = AUV_bow_vertical    * fabs(AUV_bow_vertical    ) / 24.0;
AUV_stern_vertical = AUV_stern_vertical * fabs(AUV_stern_vertical) / 24.0;
AUV_bow_lateral     = AUV_bow_lateral     * fabs(AUV_bow_lateral     ) / 24.0;
AUV_stern_lateral   = AUV_stern_lateral   * fabs(AUV_stern_lateral ) / 24.0;

// slots 4..7 as yet unused
```

```
    // cout << "viewer: Articulation parameter 1:  thrusters"
    //      << endl;
    // cout << "AUV_bow_vertical    = " << AUV_bow_vertical    << endl;
    // cout << "AUV_stern_vertical  = " << AUV_stern_vertical  << endl;
    // cout << "AUV_bow_lateral     = " << AUV_bow_lateral     << endl;
    // cout << "AUV_stern_lateral   = " << AUV_stern_lateral   << endl;

    APNptr = APNptr->next_articulat_params; // next articulated parameter

    AUV_ST1000_bearing  = APNptr->articulat_params.parameter_value [0] * 10 +
                          APNptr->articulat_params.parameter_value [1];
    AUV_ST1000_range    = APNptr->articulat_params.parameter_value [2] / 4.0;
    AUV_ST1000_strength = APNptr->articulat_params.parameter_value [3];
    AUV_ST725_bearing   = APNptr->articulat_params.parameter_value [4] * 10 +
                          APNptr->articulat_params.parameter_value [5];
    AUV_ST725_range     = APNptr->articulat_params.parameter_value [6] / 4.0;
    AUV_ST725_strength  = APNptr->articulat_params.parameter_value [7];

    // cout << "viewer: Articulation parameter 2:  sonar" << endl;
    // cout << "AUV_ST1000_bearing  = " << AUV_ST1000_bearing  << endl;
    // cout << "AUV_ST1000_range    = " << AUV_ST1000_range    << endl;
    // cout << "AUV_ST1000_strength = " << AUV_ST1000_strength << endl;
    // cout << "AUV_ST725_bearing   = " << AUV_ST725_bearing   << endl;
    // cout << "AUV_ST725_range     = " << AUV_ST725_range     << endl;
    // cout << "AUV_ST725_strength  = " << AUV_ST725_strength  << endl;

    // Print hostname of PDU (revision in network.round version)
    // hp = gethostbyaddr((char *) &inaddr, sizeof(struct in_addr), AF_INET);
    // cout << "viewer: Host name:  " << hp->h_name << endl;

    // don't forget freePDU or get a memory leak!
    // articulated parameters are also freed

    freePDU (local_PDU);

    // cout << "viewer: freePDU (local_PDU) called for this PDU" << endl;

} // end while infinite loop

// cout << "viewer: DIS net_read portion complete, now update scene graph."
//      << endl;

current_clock      = clock ();
delta_clock        = current_clock - time_of_PDU_receipt;
delta_time         = (double) delta_clock / CLOCKS_PER_SEC;

// cout << "viewer: current_clock = "                  << current_clock
//      << ", time_stamp_of_current_PDU = " << time_stamp_of_current_PDU
//      << ", time_of_PDU_receipt = "       << time_of_PDU_receipt
//      << ", PDU delta_time = "            << delta_time            << endl;

if ((delta_time >= 0.0) && (delta_time <= 5.0)) // update positions, postures
{
    delta_x      = AUV_x_dot     * delta_time;
    delta_y      = AUV_y_dot     * delta_time;
    delta_z      = AUV_z_dot     * delta_time;

    delta_phi    = AUV_phi_dot   * delta_time;
    delta_theta  = AUV_theta_dot * delta_time;
    delta_psi    = AUV_psi_dot   * delta_time;

    // cout << "viewer: DeadReckon_Callback: "
    //      << " PDU delta_time = " << delta_time                  << endl;
    // cout << " AUV_phi         = " << degrees (AUV_phi)
    //      << ", AUV_phi_dot    = " << degrees (AUV_phi_dot)     << endl;
    // cout << " AUV_theta       = " << degrees (AUV_theta)
    //      << ", AUV_theta_dot  = " << degrees (AUV_theta_dot)   << endl;
```

```
// cout << "  delta_x        = " << delta_x                   << endl;
// cout << "  delta_y        = " << delta_y                   << endl;
// cout << "  delta_z        = " << delta_z                   << endl;
// cout << endl;

// save current position for next time the camera is repositioned
AUV_x_prior = AUV_x;
AUV_y_prior = AUV_y;
AUV_z_prior = AUV_z;

// graphics rendering problems:  psi, rudders are opposite

// update overall AUV posture (both position & orientation)
AUV_position_node->translation.setValue ( AUV_x     + delta_x,
                                        - (AUV_y     + delta_y),
                                        - (AUV_z     + delta_z));
rotate_AUV_z->angle.setValue          ( - (AUV_psi   + delta_psi));
rotate_AUV_y->angle.setValue          ( - (AUV_theta + delta_theta));
rotate_AUV_x->angle.setValue              (AUV_phi   + delta_phi);

// update AUV rudder & plane orientations

rotate_AUV_bow_rudders->angle.setValue    (   radians(AUV_delta_rudder));
rotate_AUV_stern_rudders->angle.setValue  ( - radians(AUV_delta_rudder));
rotate_AUV_bow_planes->angle.setValue     ( - radians(AUV_delta_planes));
rotate_AUV_stern_planes->angle.setValue   (   radians(AUV_delta_planes));

// cout << "AUV_delta_rudder = " << AUV_delta_rudder
//       << ", radians (AUV_delta_rudder) = "
//       << radians (AUV_delta_rudder)
//       << endl;
// cout << "AUV_delta_planes= " << AUV_delta_planes
//       << ", radians (AUV_delta_planes) = "
//       << radians (AUV_delta_planes)
//       << endl;

if (fabs(AUV_stbd_rpm -
       (conePropellerWakeStbd->height.getValue() * 700.0/24.0)) > 10.0)
// ensure needed
{
   conePropellerWakeStbd->height       =       AUV_stbd_rpm  /700.0*24.0;
   conePropellerWakeStbd->bottomRadius = fabs (AUV_stbd_rpm) /700.0* 6.0;
}
if (fabs(AUV_port_rpm  -
      (conePropellerWakePort->height.getValue() * 700.0 / 24.0)) > 10.0)
// ensure needed
{
   conePropellerWakePort->height       =       AUV_port_rpm  /700.0*24.0;
   conePropellerWakePort->bottomRadius = fabs (AUV_port_rpm) /700.0* 6.0;
}
if (fabs(AUV_bow_vertical-(thrusterWakeFV->height.getValue()/2.0)) > 1.0)
// ensure needed
{
   thrusterWakeFV->height       = - AUV_bow_vertical * 2.0;
   thrusterWakeFV->bottomRadius =   AUV_bow_vertical / 4.0;
   if   (AUV_bow_vertical < 0.0)
        // bottomsideBow, negative volts push up   (negative direction)
        whichWakeFV->whichChild = 1;
        //    topsideBow, positive volts push down (positive direction)
   else whichWakeFV->whichChild = 0;

   topsideBow->    translation.setValue(0,    TOPHALF+AUV_bow_vertical,0);
   bottomsideBow->translation.setValue(0, BOTTOMHALF+AUV_bow_vertical,0);
}
if (fabs(AUV_stern_vertical-(thrusterWakeAV->height.getValue()/2.0))
        > 1.0)
// ensure needed
```

```
    {
       thrusterWakeAV->height        = - AUV_stern_vertical * 2.0;
       thrusterWakeAV->bottomRadius =   AUV_stern_vertical / 4.0;
       if    (AUV_stern_vertical < 0.0)
             // bottomsideStern, negative volts push up  (negative direction)
             whichWakeAV->whichChild = 1;
             //    topsideStern, positive volts push down(positive direction)
       else  whichWakeAV->whichChild = 0;

      topsideStern->  translation.setValue(0,   TOPHALF+AUV_stern_vertical,0);
      bottomsideStern->translation.setValue(0,BOTTOMHALF+AUV_stern_vertical,0);
    }
    if (fabs(AUV_bow_lateral-(thrusterWakeFH->height.getValue() /2.0)) > 1.0)
    // ensure needed
    {
       thrusterWakeFH->height        = - AUV_bow_lateral * 2.0;
       thrusterWakeFH->bottomRadius =   AUV_bow_lateral / 4.0;
       if    (AUV_bow_lateral < 0.0)
             // rightsideBow, negative volts push left  (negative direction)
             whichWakeFH->whichChild = 1;
             //  leftsideBow, positive volts push right (positive direction)
       else  whichWakeFH->whichChild = 0;

       leftsideBow-> translation.setValue ( 0, LEFT_HALF+AUV_bow_lateral, 0);
       rightsideBow->translation.setValue ( 0, RIGHTHALF+AUV_bow_lateral, 0);
    }
    if (fabs(AUV_stern_lateral-(thrusterWakeAH->height.getValue()/2.0))
            > 1.0)
    // ensure needed
    {
       thrusterWakeAH->height        = - AUV_stern_lateral * 2.0;
       thrusterWakeAH->bottomRadius =   AUV_stern_lateral / 4.0;
       if    (AUV_stern_lateral < 0.0)
             // rightsideStern, negative volts push left (negative direction)
             whichWakeAH->whichChild = 1;
             //  leftsideStern, positive volts push right(positive direction)
       else  whichWakeAH->whichChild = 0;

       leftsideStern-> translation.setValue(0,LEFT_HALF+AUV_stern_lateral,0);
       rightsideStern->translation.setValue(0,RIGHTHALF+AUV_stern_lateral,0);
    }
    if (fabs(AUV_ST1000_range - coneSonarST1000->height.getValue()) > 0.0)
    // ensure needed
    {
       coneSonarST1000->height        = fabs (AUV_ST1000_range);
       coneSonarST1000->bottomRadius = fabs (AUV_ST1000_range) / 60.0;
                                      // 1 degree
       xfConeSonarST1000->translation.setValue (  0.0,
                                               - (AUV_ST1000_range / 2.0),
                                                 4.0 / 12.0 );
    }
}
else if (PDU_overdue == FALSE) // update scene graph, reset vehicle position
{
    PDU_overdue = TRUE; // over 5 seconds elapsed since last PDU

    // restore latest valid AUV posture (both position & orientation)
    AUV_position_node->translation.setValue (  AUV_x,
                                             - AUV_y,
                                             - AUV_z);

    rotate_AUV_z->angle.setValue           ( - AUV_psi);
    rotate_AUV_y->angle.setValue           ( - AUV_theta);
    rotate_AUV_x->angle.setValue           (   AUV_phi);

    // thrusters
    AUV_bow_vertical    = 0.0;
    AUV_stern_vertical  = 0.0;
```

```
AUV_bow_lateral      = 0.0;
AUV_stern_lateral    = 0.0;

thrusterWakeFV->height        = 0.0;
thrusterWakeFV->bottomRadius = 0.0;
thrusterWakeAV->height        = 0.0;
thrusterWakeAV->bottomRadius = 0.0;
thrusterWakeFH->height        = 0.0;
thrusterWakeFH->bottomRadius = 0.0;
thrusterWakeAH->height        = 0.0;
thrusterWakeAH->bottomRadius = 0.0;

AUV_ST1000_bearing  = 0;
AUV_ST1000_range    = 0.0;
AUV_ST1000_strength = 0;
AUV_ST725_bearing   = 0;
AUV_ST725_range     = 0.0;
AUV_ST725_strength  = 0;

coneSonarST1000->height       = fabs(AUV_ST1000_range);
coneSonarST1000->bottomRadius = fabs(AUV_ST1000_range)/60.0; // 1 degree

AUV_port_rpm = 0.0;
AUV_stbd_rpm = 0.0;

conePropellerWakePort->height       = 0.0;
conePropellerWakePort->bottomRadius = 0.0;
conePropellerWakeStbd->height       = 0.0;
conePropellerWakeStbd->bottomRadius = 0.0;

cout << "viewer: DeadReckon_Callback: "
     << "PDU delta_time = " << delta_time << ","
     << endl;
cout << "viewer position/posture reset to last received PDU." << endl;
cout << endl;
cout << flush;
}
// globals
  priorAUVPosition = SbVec3f (AUV_x_prior, AUV_y_prior, AUV_z_prior);
currentAUVPosition = SbVec3f (AUV_x,       AUV_y,       AUV_z);
aheadOfAUVPosition = currentAUVPosition +
                     SbVec3f (10.0*cos (AUV_psi), 10.0*sin(AUV_psi),2.0);

switch ( whichCamera ) // reposition appropriate camera as needed
{
case CAMERA_FREE:
     priorCameraPosition  = PerspectiveCameraFree->position.getValue ();
     priorCameraOffset    = priorCameraPosition - priorAUVPosition;
     currentCameraPosition= priorCameraPosition;
     PerspectiveCameraFree->orientation.getValue(orientationRotationAxis,
                                                  orientationRotationAngle);
     break;

case CAMERA_TO_AUV: // retain camera pos'n relative to new AUV position
     priorCameraPosition = PerspectiveCameraToAUV->position.getValue ();
     priorCameraOffset   = priorCameraPosition - priorAUVPosition;
// verify here
   currentCameraPosition =                         ( currentAUVPosition
                                                    + standardCameraOffset );
   PerspectiveCameraToAUV->position.setValue       ( currentAUVPosition
                                                    + standardCameraOffset );
   PerspectiveCameraToAUV->pointAt                 (currentAUVPosition );
   PerspectiveCameraToAUV->focalDistance.setValue(standardCameraFocalDistance);
   PerspectiveCameraToAUV->orientation.getValue    (orientationRotationAxis,
                                                    orientationRotationAngle );
     break;
```

```
        case CAMERA_FROM_AUV: // retain camera position looking out from AUV pos.
             priorCameraPosition = PerspectiveCameraFromAUV->position.getValue ();
             priorCameraOffset   = priorCameraPosition - priorAUVPosition;
// verify here
    currentCameraPosition =                        ( currentAUVPosition );
    PerspectiveCameraFromAUV->position.setValue    ( currentAUVPosition );
    PerspectiveCameraFromAUV->pointAt              ( currentAUVPosition
                                                    +standardCameraOffset );
    PerspectiveCameraFromAUV->focalDistance.setValue(standardCameraFocalDistance);
    PerspectiveCameraFromAUV->orientation.getValue  (orientationRotationAxis,
                                                     orientationRotationAngle );
        break;

        } // end switch ( whichCamera )

        // print out all camera parameters
        cout << endl;
        cout << "       AUV_position     = <" << (AUV_x)
             << ", "                          << (AUV_y)
             << ", "                          << (AUV_z)             << "> "
             << endl;
        cout << "delta_AUV_position      = <" << (AUV_x - AUV_x_prior)
             << ", "                          << (AUV_y - AUV_y_prior)
             << ", "                          << (AUV_z - AUV_z_prior)  << "> "
             << endl;
        cout << "delta_CameraPosition  ="
             << " <" << (currentCameraPosition [0] - priorCameraPosition [0])
             << ", " << (currentCameraPosition [1] - priorCameraPosition [1])
             << ", " << (currentCameraPosition [2] - priorCameraPosition [2])
             << "> " << endl;
        cout <<        "priorCameraPosition    ="
             << " <" << priorCameraPosition [0]
             << ", " << priorCameraPosition [1]
             << ", " << priorCameraPosition [2] << ">" << endl;
        cout <<        "priorCameraOffset      ="
             << " <" << priorCameraOffset [0]
             << ", " << priorCameraOffset [1]
             << ", " << priorCameraOffset [2] << ">" << endl;
        cout <<        "currentCameraPosition ="
             << " <" << currentCameraPosition [0]
             << ", " << currentCameraPosition [1]
             << ", " << currentCameraPosition [2] << ">" << endl;
        cout <<        "orientationRotation    ="
             << " <" << orientationRotationAxis [0]
             << ", " << orientationRotationAxis [1]
             << ", " << orientationRotationAxis [2]
             << ", " << degrees (orientationRotationAngle)    << ">" << endl;

   // cout << "viewer:  end of DIS_Redraw_Callback ()" << endl;

   return;
}
//----------------------------------------------------------------------------//

// called on an exit condition via a call to atexit (DIS_net_close) in main

static void DIS_net_close ()
{
   cout << "viewer: DIS_net_close ();" << endl;
   net_close ();
   DIS_port_open = FALSE;
}
//----------------------------------------------------------------------------//


////////////////////////////////////////////////////////////////////////////////
```

```
//
// This is called by the Color Editor whenever the color
// has changed. The userData is set by main() in the call
// to SoXtColorEditor::addColorChangedCallback.
//
void
colorEditorCB( void *userData, const SbColor *rgbCallbackData )
{
   SoXtRenderArea *renderArea = (SoXtRenderArea *) userData;
   renderArea->setBackgroundColor( *rgbCallbackData );
}

//////////////////////////////////////////////////////////////////////////////

SoSeparator * readFile(const char *filename) // Inventor Mentor p. 284
{
   // Open the input file
   SoInput mySceneInput;
   if (!mySceneInput.openFile(filename))
   {
      cout << "Cannot open file " << filename << endl;
      return NULL;
   }
   // Read the whole file into the database
   SoSeparator * myGraph = SoDB::readAll(&mySceneInput);
   if (myGraph == NULL)
   {
      cout << "Problem reading file " << filename << endl;
      return NULL;
   }
   mySceneInput.closeFile ();
   return myGraph;
}

//////////////////////////////////////////////////////////////////////////////

void initialize_globals ()
{
   // multicast port & group
   strncpy (port,  "3111",   4);
   strncpy (group, "224.2.121.93", 12);

   // 3111 is npsnet 'standard' port
   // #define DEFAULT_PORT  = "3111";
   // #define DEFAULT_GROUP = "224.2.121.93";

   // initialize materials
   silver-> ambientColor.setValue ( .2, .2, .2 );
   silver-> diffuseColor.setValue ( .6, .6, .6 );
   silver->specularColor.setValue ( .5, .5, .5 );
   silver->shininess = .5;

   gold-> ambientColor.setValue ( .4, .2, .0 );
   gold-> diffuseColor.setValue ( .9, .5, .0 );
   gold->specularColor.setValue ( .7, .7, .0 );
   gold->shininess = .6;

   brass-> ambientColor.setValue ( 0.329412, 0.223529, 0.027451 );
   brass-> diffuseColor.setValue ( 0.780392, 0.568627, 0.113725 );
   brass->specularColor.setValue ( 0.992157, 0.941176, 0.807843 );
   brass->shininess = 0.21794872;

   chrome-> ambientColor.setValue ( 0.25, 0.25, 0.25 );
   chrome-> diffuseColor.setValue ( 0.4,  0.4,  0.4 );
   chrome->specularColor.setValue ( 0.774597, 0.774597, 0.774597 );
   chrome->shininess = 0.6;
```

```
    npsblue-> ambientColor.setValue ( 0.0, 0.0, 1.0 );
    npsblue-> diffuseColor.setValue ( 0.0, 0.0, 0.8 );
    npsblue->specularColor.setValue ( 0.0, 0.2, 1.0 );
    npsblue->shininess = 0.8;

    seagreen-> ambientColor.setValue ( 0.0, 0.5, 0.0 );
    seagreen-> diffuseColor.setValue ( 0.0, 0.5, 0.0 );
    seagreen->specularColor.setValue ( 0.0, 0.5, 0.0 );
    seagreen->shininess = 0.0;

    darkgreen-> ambientColor.setValue ( 0.15, 0.20, 0.15 );
    darkgreen-> diffuseColor.setValue ( 0.15, 0.20, 0.15 );
    darkgreen->specularColor.setValue ( 0.15, 0.20, 0.15 );
    darkgreen->shininess = 0.0;

    sonar_red-> ambientColor.setValue ( 1.0, 0.0, 0.15 );
    sonar_red-> diffuseColor.setValue ( 1.0, 0.0, 0.15 );
    sonar_red->specularColor.setValue ( 1.0, 0.0, 0.15 );
    sonar_red->shininess = 0.0;

    return;
} // end initialize_globals ()

///////////////////////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////////////////////

void parse_command_line_flags (int argc, char ** argv)
                              // command line arguments
{
   int index, i;

   // cout << "[parse_command_line_flags start: # arguments = " << argc << "]"
   //      << endl;
   // cout << "[ " ;
   // for (i = 0; i < argc; i++) cout << argv [i] << " ";
   // cout << "]" << endl;

   for (i = 1; i < argc; i++)
   {
      for (index = 0; index <= strlen (argv[i]); index++)     // uppercase
           argv[i] [index] = toupper (argv[i] [index]);

      if      ((strcmp (argv[i],   "PORT") == 0) ||
               (strcmp (argv[i],  "-PORT") == 0) ||
               (strcmp (argv[i],      "P") == 0) ||
               (strcmp (argv[i],     "-P") == 0))
      {
          if   ( i+1 >= argc )
               cout << "Insufficient parameters for PORT" << endl;
          else
          {
               cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
               strcpy (port, argv[i+1]) ;
               i++;
          }
      }
      else if ((strcmp (argv[i],    "GROUP") == 0) ||
               (strcmp (argv[i],   "-GROUP") == 0) ||
               (strcmp (argv[i],        "G") == 0) ||
               (strcmp (argv[i],       "-G") == 0) ||
               (strcmp (argv[i],  "ADDRESS") == 0) ||
               (strcmp (argv[i], "-ADDRESS") == 0) ||
               (strcmp (argv[i],        "A") == 0) ||
               (strcmp (argv[i],       "-A") == 0))
      {
          if   ( i+1 >= argc )
               cout << "Insufficient parameters for GROUP ADDRESS" << endl;
```

```
        else
        {
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
            strcpy (group, argv[i+1]);
            i++;
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
        }
}
else if ((strcmp (argv[i],  "PRINTDIALOG") == 0) ||
         (strcmp (argv[i], "-PRINTDIALOG") == 0))
{
        PRINTDIALOG = TRUE;
        cout << "[" << argv[i]  << "]" << endl;
}
else if ((strcmp (argv[i],  "TEXTURE")    == 0) ||
         (strcmp (argv[i], "-TEXTURE")    == 0) ||
         (strcmp (argv[i],  "TEXTURE-ON") == 0) ||
         (strcmp (argv[i], "-TEXTURE-ON") == 0) ||
         (strcmp (argv[i],  "T"         ) == 0) ||
         (strcmp (argv[i], "-T"         ) == 0))
{
        TEXTURE = TRUE;
        cout << "[" << argv[i]  << "]" << endl;
}
else if ((strcmp (argv[i],  "NOTEXTURE")   == 0) ||
         (strcmp (argv[i], "-NOTEXTURE")   == 0) ||
         (strcmp (argv[i],  "TEXTURE-OFF") == 0) ||
         (strcmp (argv[i], "-TEXTURE-OFF") == 0))
{
        TEXTURE = FALSE;
        cout << "[" << argv[i]  << "]" << endl;
}
else if ((strcmp (argv[i],  "FILE") == 0) ||
         (strcmp (argv[i], "-FILE") == 0) ||
         (strcmp (argv[i],  "F")    == 0) ||
         (strcmp (argv[i], "-F")    == 0))
{
        if  ( i+1 >= argc )
            cout << "Insufficient parameters for FILE filename.iv"
                 << endl;
        else
        {
            cout << "[" << argv[i] << " " << argv[i+1] << "]" << endl;
            command_line_node = new SoSeparator;
            command_line_node = readFile (argv[i+1]);
            root->    addChild ( command_line_node );
            i++;
        }
}
else if ((strcmp (argv[i],  "NOPRINTDIALOG") == 0) ||
         (strcmp (argv[i], "-NOPRINTDIALOG") == 0))
{
        PRINTDIALOG = FALSE;
        cout << "[" << argv[i]  << "]" << endl;
}
else if ((strcmp (argv[i],  "BACKGROUNDCOLORDIALOG") == 0) ||
         (strcmp (argv[i], "-BACKGROUNDCOLORDIALOG") == 0))
{
        BACKGROUNDCOLORDIALOG = TRUE;
        cout << "[" << argv[i]  << "]" << endl;
}
else if ((strcmp (argv[i],  "NOBACKGROUNDCOLORDIALOG") == 0) ||
         (strcmp (argv[i], "-NOBACKGROUNDCOLORDIALOG") == 0))
{
        BACKGROUNDCOLORDIALOG = FALSE;
        cout << "[" << argv[i]  << "]" << endl;
}
```

```
        else cout << "Unrecognized command line parameter:  '" << argv[i]
                  << "', ignored." << endl;

    }  // end for loop through command line parameters

    // cout << "[parse_command_line_flags complete]" << endl;

    return;

} // end parse_command_line_flags ()

//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////

 main ( int argc, char ** argv )
 {
    // Initialize Inventor and Xt - these steps MUST be first calls
    // in main, without exception, or a mystery crash results.
    Widget ViewerWindowWidget = SoXt::init(argv[0]);
    if ( ViewerWindowWidget == NULL )
    {
       cout << "viewer:  ViewerWindowWidget == NULL on startup, exiting."
            << endl;
       exit( 1 );
    }
    cout << "viewer:  ViewerWindowWidget added" << endl;

    initialize_globals ();

    parse_command_line_flags (argc, argv);

    // port and group can change by command line switches
    cout << "multicast port = "  << port
         << ", address group = " << group << endl;

    cout << "creating the scene graph:" << endl;

    root = new SoSeparator;
    root->ref();
    cout << "root added" << endl;

    // correct for different coordinate system - not yet working
    // SoRotationXYZ  * coordinateSystemFlip = new SoRotationXYZ;
    // coordinateSystemFlip->angle.setValue ( M_PI );
    // coordinateSystemFlip-> axis.setValue ( SoRotationXYZ::X );
    // root->addChild( coordinateSystemFlip );

    SoRotationXYZ *ro0x = new SoRotationXYZ;
    ro0x->angle.setValue ( 0.0 );
    ro0x-> axis.setValue (SoRotationXYZ::X);

    SoRotationXYZ *ro90x = new SoRotationXYZ;
    ro90x->angle.setValue (M_PI / 2.0);
    ro90x-> axis.setValue (SoRotationXYZ::X);

    SoRotationXYZ *ro270x = new SoRotationXYZ;
    ro270x->angle.setValue (3.0 * M_PI / 2.0);
    ro270x-> axis.setValue (SoRotationXYZ::X);

    PerspectiveCameraFree      = new SoPerspectiveCamera;
    PerspectiveCameraToAUV     = new SoPerspectiveCamera;
    PerspectiveCameraFromAUV   = new SoPerspectiveCamera;

    // can't put a group or separator above camera
    // cameras using pointAt are 90 degrees twisted askew
    // Create a camera SoSwitch node
```

```
    SoSwitch * whichCameraSwitch = new SoSwitch;
    root->addChild ( whichCameraSwitch );
    whichCameraSwitch->addChild ( PerspectiveCameraFree     );
    whichCameraSwitch->addChild ( PerspectiveCameraToAUV    );
    whichCameraSwitch->addChild ( PerspectiveCameraFromAUV );
    // default whichCamera defined at top
    whichCameraSwitch->whichChild = whichCamera;

    // Create a camera rotation correction SoSwitch node
    SoSwitch * whichCameraCorrectionSwitch = new SoSwitch;
    root->addChild ( whichCameraCorrectionSwitch );
    whichCameraCorrectionSwitch->addChild ( ro0x  );
    whichCameraCorrectionSwitch->addChild ( ro270x );
    whichCameraCorrectionSwitch->addChild ( ro270x );
    whichCameraCorrectionSwitch->whichChild = whichCamera;

    root->addChild( new SoPointLight );

    SoTransform * xfLight2 = new SoTransform;
    xfLight2->translation.setValue(0.0, 0.0, - 30.0);
    root->addChild( xfLight2 );
    root->addChild( new SoPointLight );
    SoTransform * xfLight3 = new SoTransform;
    xfLight3->translation.setValue(0.0, 0.0,   30.0);
    root->addChild( xfLight3 );
    cout << "2 point lights added " << endl;

    SoXtRenderArea    * myRenderArea = new SoXtRenderArea (ViewerWindowWidget);
    SbViewportRegion    myRegion (myRenderArea->getSize ());

    currentAUVPosition = SbVec3f (AUV_x, AUV_y, AUV_z);     // globals
    aheadOfAUVPosition = currentAUVPosition +
                         SbVec3f (10.0 * cos (AUV_psi), 10.0 * sin (AUV_psi), 0.0);
                         // global

    // Free (unmodified) Camera
    PerspectiveCameraFree->viewAll (root, myRegion, 1.0);  // global
    PerspectiveCameraFree->aspectRatio.setValue (SO_ASPECT_VIDEO);

    // Camera that keeps AUV in center
    PerspectiveCameraToAUV->viewAll (root, myRegion, 1.0);  // global
    PerspectiveCameraToAUV->aspectRatio.setValue (SO_ASPECT_VIDEO);
    PerspectiveCameraToAUV->position.setValue
                                      ( currentAUVPosition + standardCameraOffset );
//    PerspectiveCameraToAUV->orientation.setValue
                                      (SbRotation (1.0, 0.0, 0.0, M_PI / 2.0 ));
    PerspectiveCameraToAUV->pointAt ( currentAUVPosition );

    // Camera that looks out from AUV in center
    PerspectiveCameraFromAUV->viewAll (root, myRegion, 1.0);  // global
    PerspectiveCameraFromAUV->aspectRatio.setValue (SO_ASPECT_VIDEO);
    PerspectiveCameraFromAUV->position.setValue ( currentAUVPosition );
//    PerspectiveCameraFromAUV->orientation.setValue
                                      (SbRotation (1.0, 0.0, 0.0, M_PI / 2.0 ));
    PerspectiveCameraFromAUV->pointAt ( currentAUVPosition + standardCameraOffset );

    unitsfeet = new SoUnits;
    unitsfeet->units.setValue ( SoUnits::FEET );
    root->addChild( unitsfeet );

    SoPickStyle * unpickablestylenode;
    SoPickStyle *   pickablestylenode;
    unpickablestylenode = new SoPickStyle;
      pickablestylenode = new SoPickStyle;
    unpickablestylenode->style.setValue ( SoPickStyle::UNPICKABLE );
      pickablestylenode->style.setValue ( SoPickStyle::BOUNDING_BOX );
```

```
// create the terrain box <<<<<<<<<<<<<<<<<<<<<<<<<<

SoCube * backgroundCube = new SoCube;
backgroundCube->width  = 400.0;
backgroundCube->height = 400.0;
backgroundCube->depth  =   0.1;

SoTransform * xfbackgroundCube = new SoTransform;
xfbackgroundCube->translation.setValue(0.0, 0.0, -50.0);

SoTexture2 * overhangTexture = new SoTexture2;
overhangTexture->filename.setValue ("overhang.rgb");

SoSeparator * sepbackgroundCube = new SoSeparator;
sepbackgroundCube->addChild( pickablestylenode );
sepbackgroundCube->addChild( unitsfeet );
sepbackgroundCube->addChild( darkgreen );
if (TEXTURE == TRUE)
    sepbackgroundCube->addChild( overhangTexture );
sepbackgroundCube->addChild( xfbackgroundCube );
sepbackgroundCube->addChild(   backgroundCube );      // remove cube here
root->addChild ( sepbackgroundCube );

// Jason with engine animation
SoSeparator * sepJason = new SoSeparator;
SoTransform *  xfJason = new SoTransform;
SoSeparator *    Jason = readFile ("Jason.iv");
xfJason->translation.setValue( -60.0, 40.0, - 25.0);  // center of pattern
sepJason->addChild ( xfJason );
// animation from Inventor Mentor 13.6.Calculator.c++
// Set up the Jason transformations
SoRotationXYZ  * danceRotate = new SoRotationXYZ;
danceRotate->angle.setValue ( 0.0 );
danceRotate-> axis.setValue ( SoRotationXYZ::Z );
sepJason->addChild ( danceRotate );
SoTranslation * danceTranslate = new SoTranslation;
sepJason->addChild ( danceTranslate );
sepJason->addChild ( Jason );
root->    addChild ( sepJason );
// Set up an engine to calculate the motion path:
// Theta is incremented using a time counter engine,
// and converted to radians using an expression in
// the calculator engine.
SoCalculator  * calcXY = new SoCalculator;
SoTimeCounter * thetaCounter = new SoTimeCounter;
thetaCounter->max = 360;
thetaCounter->step = 1;
thetaCounter->frequency = 1 / 180.0; // 180 seconds for a full cycle
calcXY->a.connectFrom(&thetaCounter->output);
calcXY->expression.set1Value(0, "ta=a*M_PI/180");  // theta (radians)
calcXY->expression.set1Value(1, "tb=15*cos(ta)");  // r, z
calcXY->expression.set1Value(2, "td=tb*cos(ta)");  // x
calcXY->expression.set1Value(3, "te=tb*sin(ta)");  // y
calcXY->expression.set1Value(4, "oA=vec3f(td,te,tb)"); // vector output A
danceTranslate->translation.connectFrom(&calcXY->oA);
calcXY->expression.set1Value(5, "ob=2*ta");          // scalar output b
danceRotate->angle.connectFrom(&calcXY->ob);

// Platform
SoSeparator * sepPlatform = new SoSeparator;
SoTransform *  xfPlatform = new SoTransform;
SoSeparator *    Platform = readFile ("Platform.iv");
xfPlatform->translation.setValue(-80.0, 50.0, -50.0);
sepPlatform->addChild ( xfPlatform );
sepPlatform->addChild( Platform );
root->addChild ( sepPlatform );
```

```
// Testtank
SoSeparator * sepTesttank = new SoSeparator;
SoTransform *  xfTesttank = new SoTransform;
SoSeparator *    Testtank = readFile ("Testtank.iv");
xfTesttank->translation.setValue( 0.0, 0.0, -50.0);
sepTesttank->addChild ( xfTesttank );
sepTesttank->addChild( Testtank );
root->addChild ( sepTesttank );

// Torpedo Tube
SoSeparator * sepTorpedoTube = new SoSeparator;
SoTransform *  xfTorpedoTube = new SoTransform;
xfTorpedoTube->translation.setValue( 30.0,  20.0, -48.00 );
SoCylinder * TorpedoTube = new SoCylinder;
TorpedoTube->radius = 21.0 / 12.0;        // 21" diameter
TorpedoTube->height = 10.0;               // 10' length
TorpedoTube->parts  = SoCylinder::SIDES;  // let's drive through!
sepTorpedoTube->addChild ( xfTorpedoTube );
sepTorpedoTube->addChild( TorpedoTube );
root->addChild ( brass );
root->addChild ( sepTorpedoTube );

SoSeparator * AUV_node = makeAUV();

root->addChild ( AUV_node ); // AUV object creation routine above

// WriteAction writes scene graph to file
FILE * auv_iv_fp = fopen ("auv.iv", "w");
SoWriteAction writeaction;
writeaction.getOutput()-> setFilePointer (auv_iv_fp);
writeaction.apply (AUV_node);
fclose (auv_iv_fp);
cout << "writeaction.apply (AUV_node) => auv.iv complete." << endl;

// WriteAction writes scene graph to file
FILE * auv_vw_iv_fp = fopen ("auv_vw.iv", "w");
writeaction.getOutput()-> setFilePointer (auv_vw_iv_fp);
writeaction.apply (root);
fclose (auv_vw_iv_fp);
cout << "writeaction.apply (root) => auv_vw.iv complete." << endl;

DIS_net_open ();
atexit (DIS_net_close);   // ensure port is reclosed on exit.  tested sat.
current_clock = clock (); // initialize

// A TimerSensor updates the object with DIS postures and performs redraws
SoTransform    * dummy_xform    = new SoTransform;
SoTimerSensor * DIS_Redraw_Sensor = new SoTimerSensor( DIS_Redraw_Callback,
                                                        dummy_xform );
DIS_Redraw_Sensor->setInterval ( 0.10 );  // seconds
DIS_Redraw_Sensor->schedule ();

// system ("rm     sounds/nps_auv.au");
// system ("www -o sounds/nps_auv.au
file://taurus.cs.nps.navy.mil/pub/auv/nps_auv.au");
// system ("www -o sounds/nps_auv.au
http://www_tios.cs.utwente.nl/say/?Naval+Postgraduate+School,Autonomous+Underwater+Veh
icle");
// system ("sfplay sounds/nps_auv.au &");

if (PRINTDIALOG == TRUE)
{
   // Print dialog widget:  Inventor training manual p. 9-9
   SoXtPrintDialog *printDialog = new SoXtPrintDialog;
   printDialog->setSceneGraph (root);
   printDialog->show   ();
}
```

```
/* coloreditor not found?!
   if (BACKGROUNDCOLORDIALOG == TRUE)
   {

      // Build the color editor in its own window
         SoXtColorEditor *color_editor = new SoXtColorEditor;
         color_editor->build();
         color_editor->setTitle( "AUV viewer background color" );
      // Add a callback for when the color changes
         color_editor->addColorChangedCallback( colorEditorCB, // the callback
                                                viewer ); // user data to be passed
         SbColor lightbluecolor( .0, .5, .75 );
         viewer->setBackgroundColor( lightbluecolor );
         color_editor->setColor    ( lightbluecolor );
         color_editor->show();      // Display the color editor
   }
*/

// Uncomment which viewer you want to use:
   SoXtExaminerViewer * viewer = new SoXtExaminerViewer;
// SoXtFlyViewer      * viewer = new SoXtFlyViewer;
// SoXtPlaneViewer    * viewer = new SoXtPlaneViewer;
// SoXtWalkViewer     * viewer = new SoXtWalkViewer;

   viewer->setSceneGraph( root );
   viewer->setTitle("NPS  AUV  Virtual  World");
   viewer->show();
// XtRealizeWidget ( ViewerWindowWidget ); // mini window junk
   SoXt::mainLoop(); // loop forever, sending events to the scene graph
}
// end of viewer.C
////////////////////////////////////////////////////////////////////////////////
```

C. *Makefile* for Object-Oriented Real-Time Graphics Viewer

```
#!smake

PROGRAM = viewer

C++FILES = viewer.C

# /usr/include/make/commondefs and /usr/include/make/commonrules
# define some useful Makefile rules.  For example, they
# defines a 'clean' target, so you can type 'make clean'
# to remove all .o files (or other files generated during
# compilation). See the file /usr/include/make/commonrules for
# documentation on what targets are supported.

include /usr/include/make/commondefs

TARGETS = $(PROGRAM)

OBJECTS = viewer.o

# Libraries to link with:

LLDLIBS =    -lInventorXt

##############################################################################
# DIS includes

INCS            = -I../DIS.mcast/h
LIB_DIR                 = -L../DIS.mcast/src
LIB             = ../DIS.mcast/src/libdis_client.a
LIB_ARG                 = -ldis_client
HDR_PATH        = ../DIS.mcast/h/
HDRS            = $(HDR_PATH)pdu.h $(HDR_PATH)disdefs.h

##############################################################################

default: $(TARGETS)

include $(COMMONRULES)

$(TARGETS): $(OBJECTS)
        CC $(C++FILES) -o $@ $(OBJECTS) $(LDFLAGS) $(LIB) $(LIB_DIR) $(INCS) \
        $(LIB_ARG)  $(LLDLIBS)


# modified from patton:/usr/share/src/Inventor/samples/clock/Makefile

# updated  2 October 94  Don Brutzman
```

V. UNDERWATER VIRTUAL WORLD HYDRODYNAMICS

A. Introduction

Structuring the model design problem was the key to comprehensible implementation. A straightforward hierarchy follows. Posture is common to all vehicles and can be represented either by Euler angle rotations, by a homogenous transformation matrix (Fu 87) (Foley, van Dam 90), or by quaternions (Cooke 92). Forces and accelerations acting upon a rigid body, if modeled, are related by dynamics equations of motion corresponding to each spatial degree of freedom. A rigid body is further subject to kinematics equations of motion which combine velocities with postures in strictly defined ways regardless of vehicle type or environmental dimensionality. A networked rigid body which communicates with other entities via DIS needs to calculate postures, optional linear and rotational velocities, and (again optional) linear accelerations (IEEE 93). Such a DIS-networked rigid body has identical capabilities regardless of vehicle type.

An entity dynamics component for a real-time networked virtual world combines the functionality of rigid bodies and DIS networking with the dynamics equations of motion (forces and accelerations) unique to a specific vehicle type. This structured hierarchy of relationships between posture representations, rigid bodies, DIS networking and dynamics equations of motion led to the general model class diagram which appears in Figure 1.

The compartment boxes within Figure 1 delineate the functionality of class components. The first compartment is class name. The second compartment indicates member data fields, which are the data structures encapsulated by the object. The third compartment indicates object methods (functions) which effectively occur instantaneously. The fourth compartment includes methods (functions) which are time-consuming, either from the perspective of simulation clock duration or actual delay due to network latency. Adapted from the Object-Oriented Simulation Pictures

(OOSPICs) design and testing methodology (Bailey 94), this diagraming approach is very useful because it simplifies presentation of key object relationships and clarifies hierarchy design. Of particularly value is the explicit specification of temporal relationships, which are critical to success in a real-time system and are often overlooked in complex system design. An example object template which adapts the OOSPICs methodology from ***MODSIM*** programming language to ***C++*** appears as Figure 2. A key for OOSPIC arrow conventions is included in Figure 3 (Bailey 94). Although ***C++*** OOSPICs are not provided for each class, inspection of specifications in the accompanying source code reveals that every class follows the structural layout presented in Figure 1.

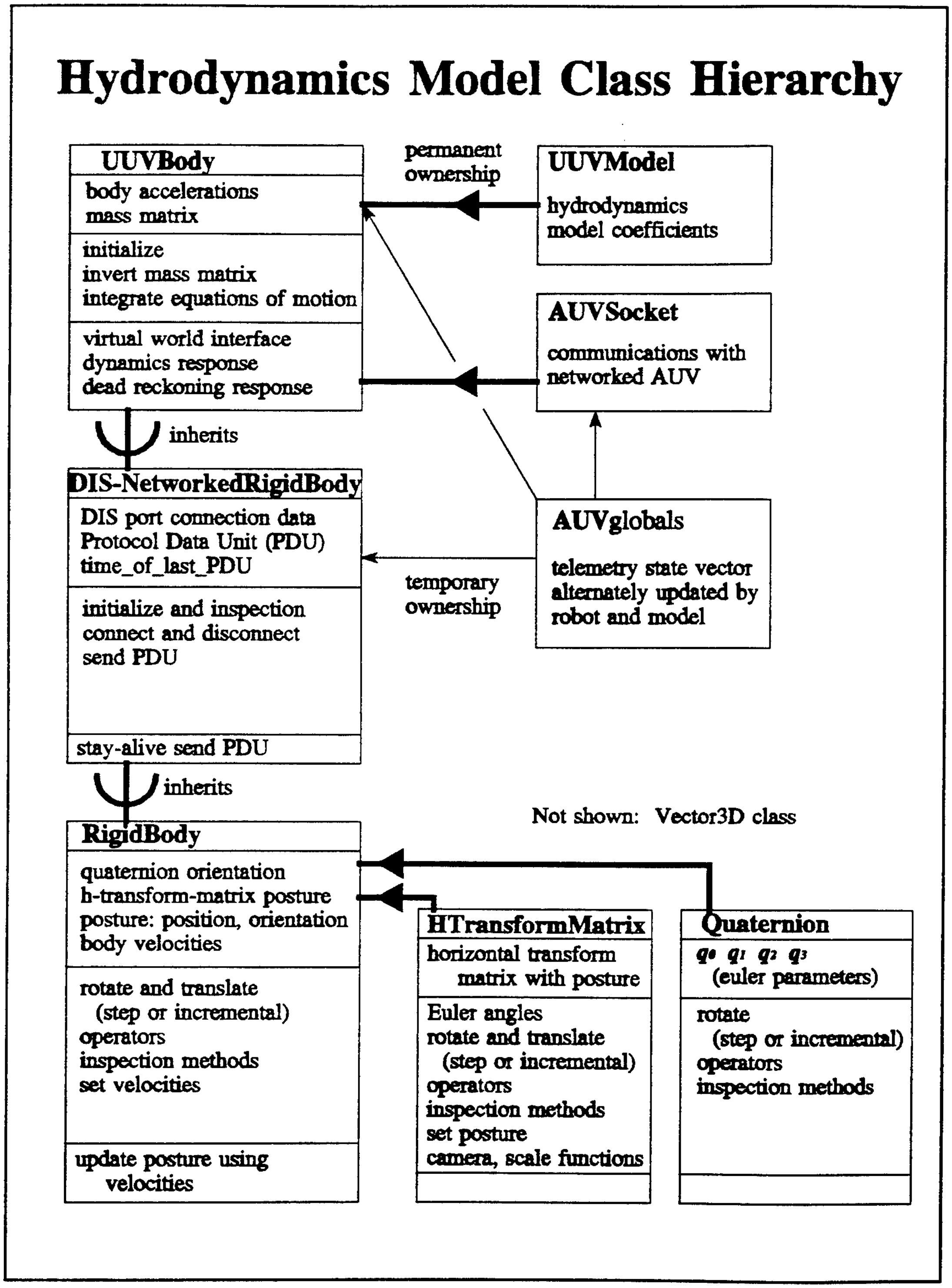


Figure 1. General real-time DIS-networked hydrodynamics model class hierarchy.

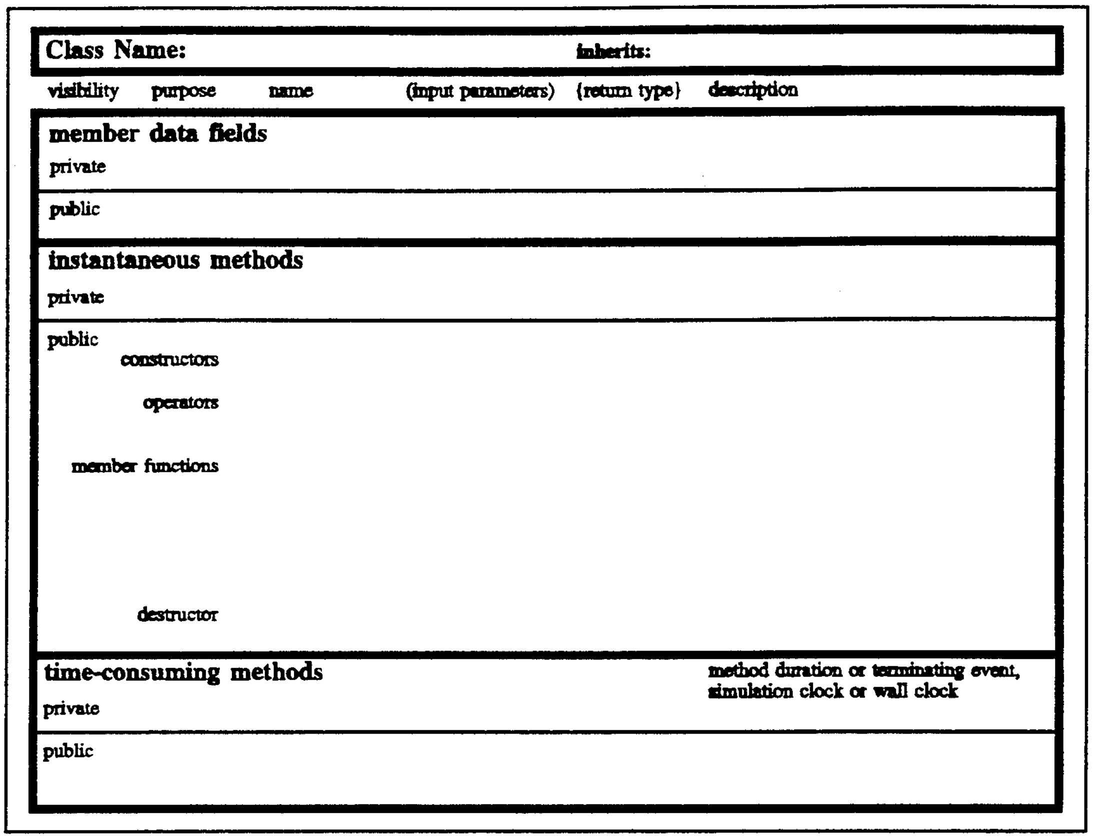


Figure 2. OOSPIC class diagram template for ***C++*** class definitions. Separation of class name, data fields, instantaneous methods and temporal methods clarifies class functionality and design.

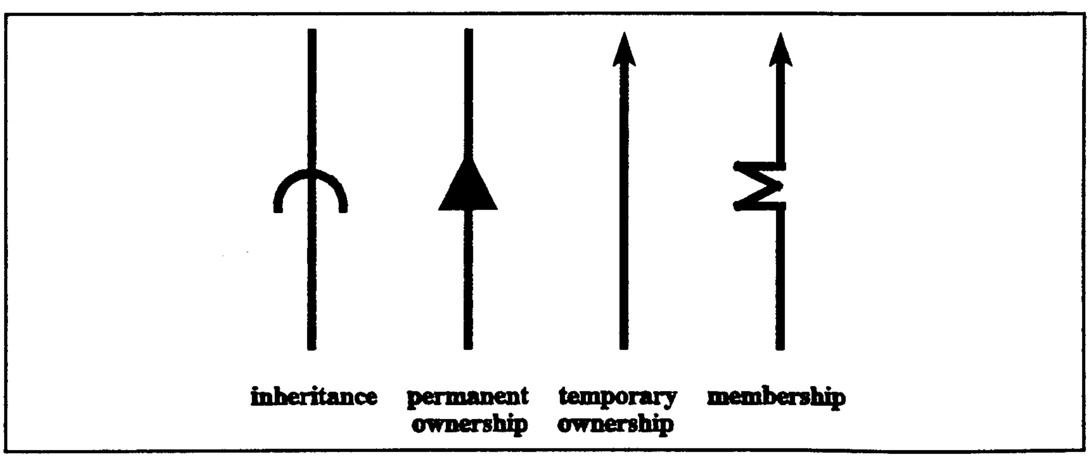


Figure 3. Object-Oriented Simulation Pictures (OOSPICs) arrow conventions.

The structure of the general real-time DIS-networked dynamics model presented here appears to be applicable to vehicles of arbitrary type. Documented source code for each member of the class hierarchy matches the equations and algorithms presented in this work, and also follows the structure appearing in Figure 2 (Brutzman 94). Future work of interest in model design includes directly porting this model to emulate the characteristics of other underwater vehicles, adapting the model to accommodate dissimilar vehicle entities, porting the model into robot software as an on-board hydrodynamics response predictor, and investigating extensions to the model to support visualization, validation and verification of model relationships against archived or live data records of actual vehicle dynamics performance.

B. *dynamics.C* Virtual World Interface and Top-Level Hydrodynamics

The *dynamics* program is the server connection to the underwater virtual world. It must be started prior to the robot *execution* program so that the robot has a virtual world to connect to. *dynamics* also provides a user interface for setting ocean currents, multicast network parameters and hydrodynamics class library test routines. The user menu appears in Figure 4.

The *dynamics* program has a text-only user interface in order to reduce processor loading, maximize portability and facilitate remote operation. The menu option to reset multicast group parameters permits starting a virtual world channel using collision-free parameters automatically chosen by the MBone *sd* session directory application. The menu option for ocean currents accepts a constant 3D vector. Extending the overall functionality of the underwater virtual world is accomplished by building hooks between new models (such as time-varying ocean currents) and the *dynamics* program. New models can either be embedded directly within *dynamics*, or can be distributed and communicate via sockets.

```
dude:/n/dude/work/brutzman/dynamics>> dynamics

-------------------------------------------------------------
Dynamics classes test selections

              L loop_test_with_execution_level ();
              M Multicast parameter input
                ttl=15, group address=224.2.121.93, port=3111
              O Ocean current vector reset
                <0, 0, 0>
              H Hmatrix/quaternion exerciser
              R Rotation of quaternion & Hmatrix using p q r
              D Defaults
              I Invert matrix test
              E dEad_reckon_test_with_execution_level
              P PDU_skip_interval change (from 1)
              T Toggle TRACE = 0
              C DIS_net_close ();
              Q Quit

Enter choice:  _
```

Figure 4. *dynamics* virtual world server: user interface.

```
//////////////////////////////////////////////////////////////////////////////////
/*
 Program:           dynamics.C

 Description:       hydrodynamics model and virtual world manager

 Author:            Don Brutzman

 Revised:           18 October 94

 System:            Irix 5.2

 Compiler:          ANSI C++

 Compilation:       irix> CC   dynamics.C -lm -w -g -o dynamics

               [or] unix> make dynamics

                      +w == Warn about all questionable constructs.

 Advisors:          Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey


 Lessons:           Need single char input without requiring <CR>

                    Don't use 'static' for member data fields or mysterious linker
                        eRrors result
```

```
                    Don't use << endl << endl more than once in a single cout call
                         or a vague warning results with an invalid line number:
                        'warning:  ostream&ostream::operator <<(ostream&(*)
                         (ostream&)) not inlined, called twice in an expression'

*/
//////////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <ctype.h>

#include "UUVBody.C"

#include <bstring.h>

//////////////////////////////////////////////////////////////////////////////////////
// global data

   char                    port  [ 6];    // pointer to MULTICAST_PORT
   char                    group [20];    // MULTICAST_GROUP;
   int                     int_ttl = 15; // time-to-live

   char                    choice = '*';  // dummy value to print menu first time

   int                     READ_MENU_CHOICE = TRUE;

//////////////////////////////////////////////////////////////////////////////////////

void parse_command_line_flags (int argc, char ** argv)
                              // command line arguments
{
   int index, i;

// 3111 is npsnet 'standard' port
   strncpy (port,  "3111",   4);
   strncpy (group, "224.2.121.93", 12);

   // cout << "[parse_command_line_flags start: # arguments = " << argc
   //      << "]" << endl;
   // cout << "[ " ;
   // for (i = 0; i < argc; i++) cout << argv [i] << " ";
   // cout << "]" << endl;

   for (i = 1; i < argc; i++)
   {
      for (index = 0; index <= strlen (argv[i]); index++)     // uppercase
           argv[i] [index] = toupper (argv[i] [index]);
   }

   // cout << "[parse_command_line_flags uppercase: # arguments = " << argc
   //      << "]" << endl;
   // cout << "[ " ;
   // for (i = 0; i < argc; i++) cout << argv [i] << " ";
   // cout << "]" << endl;

   for (i = 1; i < argc; i++)
   {
      if      ((strcmp (argv[i],  "TTL") == 0) ||
               (strcmp (argv[i], "-TTL") == 0) ||
               (strcmp (argv[i],    "T") == 0) ||
               (strcmp (argv[i],   "-T") == 0))
      {
            if  ( i+1 >= argc )
                 cout << "Insufficient parameters for TTL" << endl;
```

```
            else
            {
                 int_ttl = atoi (argv[i+1]);
                 cout << "dynamics: ttl reset to " << int_ttl
                      << endl;
                 i++;
            }

       }
       else if ((strcmp (argv[i],  "PORT") == 0) ||
                (strcmp (argv[i], "-PORT") == 0) ||
                (strcmp (argv[i],     "P") == 0) ||
                (strcmp (argv[i],    "-P") == 0))
       {
            if  ( i+1 >= argc )
                 cout << "Insufficient parameters for PORT" << endl;
            else
            {
                 bzero  (port,  6);
                 strcpy (port, argv[i+1]);
                 cout << "dynamics: port reset to " << port
                      << endl;
                 i++;
            }

       }
       else if ((strcmp (argv[i],     "GROUP") == 0) ||
                (strcmp (argv[i],    "-GROUP") == 0) ||
                (strcmp (argv[i],         "G") == 0) ||
                (strcmp (argv[i],        "-G") == 0) ||
                (strcmp (argv[i],   "ADDRESS") == 0) ||
                (strcmp (argv[i],  "-ADDRESS") == 0) ||
                (strcmp (argv[i],         "A") == 0) ||
                (strcmp (argv[i],        "-A") == 0))
       {
            if  ( i+1 >= argc )
                 cout << "Insufficient parameters for GROUP ADDRESS" << endl;
            else
            {
                 bzero  (group,  20);
                 strcpy (group, argv[i+1]) ;
                 cout << "dynamics: group address reset to " << group
                      << endl;
                 i++;
            }
       }
       else if ((strcmp (argv[i],    "LOOP") == 0) ||
                (strcmp (argv[i],   "-LOOP") == 0) ||
                (strcmp (argv[i],       "L") == 0) ||
                (strcmp (argv[i],      "-L") == 0))
       {
            choice = 'L';
            READ_MENU_CHOICE = FALSE;
            cout << "dynamics: Loop test with execution level set " << endl;
       }
       else if ((strcmp (argv[i],  "TRACE") == 0) ||
                (strcmp (argv[i], "-TRACE") == 0))
       {
                 TRACE = TRUE;
                 cout << "dynamics:  TRACE is now on" << endl;
       }
       else
       {
            cout << "dynamics:  unrecognized command flag " << argv[i] << endl;
       }

  }  // end for loop through command line parameters
```

```
// cout << "[parse_command_line_flags complete]" << endl;

return;

} // end parse_command_line_flags ()

/////////////////////////////////////////////////////////////////////////////////////

void main ( int argc, char ** argv )
{
  double                a1,   b1,     c1,
                        x1,   y1,     z1,
                        p,    q,      r,        delta_time;

  int                   done = FALSE;
  int                   index, repetitions;

  Vector3D              vector1, vector2;

  Quaternion            quaternion1, quaternion2;

  Hmatrix               hmatrix1, hmatrix2;

  RigidBody             rigidbody1;

  DISNetworkedRigidBody disnetworkedrigidbody1;

  UUVBody               uuvbody1;

// main start

  parse_command_line_flags (argc, argv);

 do
 {
   if ( ! isspace (choice))
   {
     cout << endl;
     cout << "------------------------------------------------------------";
     cout << endl;
     cout << "Dynamics classes test selections"                          << endl;
     cout << "                 L loop_test_with_execution_level ();"     << endl;
     cout << "                 M Multicast parameter input"              << endl;
     cout << "                   ttl=" << int_ttl << ", group address=" << group
          <<                  ", port=" << port                          << endl;
     cout << "                 O Ocean current vector reset "            << endl;
     cout << "                   <" << AUV_oceancurrent_u << ", "
                                   << AUV_oceancurrent_v << ", "
                                   << AUV_oceancurrent_w << "> "        << endl;
     cout << "                 H Hmatrix/quaternion exerciser"           << endl;
     cout << "                 R Rotation of quaternion & Hmatrix using p q r "
                                                                         << endl;
     cout << "                 D Defaults"                               << endl;
     cout << "                 I Invert matrix test"                     << endl;
     cout << "                 E dEad_reckon_test_with_execution_level" << endl;
     cout << "                 P PDU_skip_interval change (from "
          <<              uuvbody1.PDU_skip_interval_value () << ")"    << endl;
     cout << "                 T Toggle TRACE = " << TRACE               << endl;
     cout << "                 C DIS_net_close ();"                      << endl;
     cout << "                 Q Quit"                                   << endl;
     cout << "Enter choice:    ";
     cout << flush;
   }

   if (READ_MENU_CHOICE == TRUE)
   {
```

```
        // still not reading a single char properly :(
        // cin.get (choice);
        cin  >> choice;
        cout << endl;
    }
    else cout << choice << endl;

    READ_MENU_CHOICE = TRUE;

    switch (choice)
    {
     case 'L':  case 'l': //-----------------------------------------------

        cout << "Loop test with execution level" << endl;

        uuvbody1.UUVBody_initialize              ();
        uuvbody1.set_ttl                         (int_ttl);
        uuvbody1.set_group                       (group);
        uuvbody1.set_port                        (port);

        cout << "dynamics: uuvbody.loop_test_with_execution_level ()" << endl;

        uuvbody1.loop_test_with_execution_level ();

        choice = '*';  // dummy value to print keyboard menu
        break;

     case 'O':  case 'o': //-----------------------------------------------

        cout << " Ocean current vector reset " << endl;
        cout << "______________________________" << endl;
        cout << endl;

        cout << "Enter AUV_oceancurrent in North direction:    ";
        cin  >> AUV_oceancurrent_u;
        cout << "  new AUV_oceancurrent_u = " << AUV_oceancurrent_u << endl;
        cout << endl;

        cout << "Enter AUV_oceancurrent in East  direction:    ";
        cin  >> AUV_oceancurrent_v;
        cout << "  new AUV_oceancurrent_v = " << AUV_oceancurrent_v << endl;
        cout << endl;

        cout << "Enter AUV_oceancurrent in Downward direction: ";
        cin  >> AUV_oceancurrent_w;
        cout << "  new AUV_oceancurrent_w = " << AUV_oceancurrent_w << endl;
        cout << endl;

        choice = '*';  // dummy value to print keyboard menu
        break;

     case 'M':  case 'm': //-----------------------------------------------

        cout << " Multicast parameter input " << endl;
        cout << "______________________________" << endl;
        cout << endl;

        cout << "Enter time to live ttl:  ";
        cin  >> int_ttl;
        cout << "  new time to live ttl = " << int_ttl << endl;
        cout << endl;

        cout << "Enter group address:     ";
        cin  >> group;
        cout << "  new group address    = " << group << endl;
        cout << endl;
```

```
    cout << "Enter port:                ";
    cin  >> port;
    cout << "  new port                = " << port << endl;
    cout << endl;

    choice = '*';  // dummy value to print keyboard menu
    break;

  case 'H':  case 'h': //------------------------------------------------

    cout << "Hmatrix/quaternion exerciser:" << endl;

    cout << "Enter euler angles in degrees for object:  ";
    cin  >> a1 >> b1 >> c1;
    cout << " angles entered = <" << a1 << ", " << b1 << ", " << c1 << ">"
         << endl;

    quaternion1 = Quaternion (radians (a1), radians (b1), radians (c1));

    cout << "quaternion1:   ";
    quaternion1.print ();
    quaternion1.normalize ();

    cout << endl;
    cout << "normalized:    " << quaternion1 << endl;

    vector2 = quaternion1.euler_angles ();
    cout << "euler angles1: " << vector2 << endl;

    cout << "phi theta psi1:" << quaternion1.phi_value   () << ", "
                            << quaternion1.theta_value () << ", "
                            << quaternion1.psi_value   () << endl;
    cout << "degrees:       " << degrees (quaternion1.phi_value   ()) << ", "
                            << degrees (quaternion1.theta_value ()) << ", "
                            << degrees (quaternion1.psi_value   ()) << endl;
    cout << endl;

    cout << "Enter vector positions for object:  ";
    cin  >> x1 >> y1 >> z1;

    vector1 = Vector3D (x1, y1, z1);
    cout << "vector1:        " << vector1 << endl;

    hmatrix1 = Hmatrix ( vector1, radians (a1), radians (b1), radians (c1));
    cout << "hmatrix1:       " << hmatrix1 << endl;

    cout << " hmatrix1 angles: " << degrees (hmatrix1.phi_value   ()) << ", "
                               << degrees (hmatrix1.theta_value ()) << ", "
                               << degrees (hmatrix1.psi_value   ()) << endl;
    choice = '*';  // dummy value to print keyboard menu
    break;

  case 'R': case'r':   //------------------------------------------------

    cout << "Rotation" << endl;
    cout << endl;
    cout << "Enter euler angles in degrees for object:       ";
    cin  >> a1 >> b1 >> c1;

    quaternion1 = Quaternion (radians (a1), radians (b1), radians (c1));

    cout << "Enter rotation rates in degrees per sec <p, q, r>:  ";
    cin  >> p >> q >> r ;
    p = radians (p);
    q = radians (q);
    r = radians (r);
```

```
    cout << "Enter repetitions and delta_time in seconds:     ";
    cin  >> repetitions >> delta_time;

    cout << "Enter vector positions for object:               ";
    cin  >> x1 >> y1 >> z1;
    vector1 = Vector3D (x1, y1, z1);

    hmatrix1 = Hmatrix (vector1, radians (a1), radians (b1), radians (c1));
    hmatrix2 = Hmatrix (vector1, radians (a1), radians (b1), radians (c1));

    for (index=0; index <= repetitions; index++)
    {
        cout << "quaternion = ";
        quaternion1.print ();
        cout << endl;
        cout << "quaternion1 angles = <"
             << degrees (quaternion1.phi_value   ()) << ", "
             << degrees (quaternion1.theta_value ()) << ", "
             << degrees (quaternion1.psi_value   ()) << "> " << endl;
        quaternion1.incremental_rotate (p, q, r, delta_time);

        cout << "   hmatrix1 angles = <" << degrees (hmatrix1.phi_value   ())
                                 << ", " << degrees (hmatrix1.theta_value ())
                                 << ", " << degrees (hmatrix1.psi_value   ())
                                 << "> " << endl;

        cout << "   hmatrix2 angles = <" << degrees (hmatrix2.phi_value   ())
                                 << ", " << degrees (hmatrix2.theta_value ())
                                 << ", " << degrees (hmatrix2.psi_value   ())
                                 << "> " << endl;

        cout << "hmatrix1 = " << hmatrix1 << endl;

     // two alternate methods can be tested:
        hmatrix1.incremental_rotation (p, q, r, delta_time);
        hmatrix2.rotate (p * delta_time, q * delta_time, r * delta_time);

        cout << "--------------------------------------------------" << endl;
        cin.get (choice); // pause
    }
    choice = '*';  // dummy value to print keyboard menu
    break;

  case 'D':  case 'd': //-----------------------------------------------

    cout << "Defaults" << endl;
    cout << endl;

    vector1     = Vector3D   ();
    quaternion1 = Quaternion ();
    hmatrix1    = Hmatrix    ();

    cout << "Using << operators:" << endl;
    cout << "Default vector3D:     " << vector1     << endl;
    cout << "Default quaternion:   " << quaternion1 << endl;
    cout << "Default Hmatrix:      " << hmatrix1;

    cout << "Using print methods:" << endl;
    cout << "Default vector3D:     ";
    vector1.print ();
    cout << endl << "Default quaternion:   ";
    quaternion1.print ();
    cout << endl << "Default Hmatrix:      ";
    hmatrix1.print_hmatrix ();

    cout << endl;
    cout << "initializations & assignment:" << endl;
```

```
  vector1      = Vector3D    (1, 2, 3);
  cout        << " Vector3D    (1, 2, 3) == " << vector1 << endl;
  quaternion1 = Quaternion (0, 0, 0);
  cout        << " Quaternion (0, 0, 0) == " << quaternion1 << endl;
  hmatrix1     = Hmatrix ( 40, 50, 60,
                           radians (0), radians (-90), radians (0) );
  cout        << " Hmatrix ( 40, 50, 60, radians(0), radians(-90),"
              << " radians(0)) " << hmatrix1 << endl;

  hmatrix1     = Hmatrix (Vector3D(40,50,60),
                          Vector3D(radians(0), radians(-90), radians(0)));
  cout        << " Hmatrix (Vector3D(40, 50, 60),"
              << " Vector3D(radians (0), radians(-90), radians(0))) "
              << hmatrix1 << endl;

  quaternion1 = Quaternion (radians (90), 0, 0) *
                Quaternion (0, 0, radians (90));
  cout << "quaternion multiplicaton test <90,0,0> * <0,0,90> =>" << endl;
  cout << " <" << degrees (quaternion1.phi_value   ()) << ", "
               << degrees (quaternion1.theta_value ()) << ", "
               << degrees (quaternion1.psi_value   ()) << "> " << endl;

  rigidbody1          = RigidBody ();
  cout << endl;
  cout << " RigidBody () defaults printed using different methods:"
       << rigidbody1;
  rigidbody1.print_rigidbody ();
  cout << endl;

  disnetworkedrigidbody1.DISNetworkedRigidBody_initialize ();
  cout << " DISNetworkedRigidBody () defaults printed using "
       << "different methods:" << disnetworkedrigidbody1;
  disnetworkedrigidbody1.print_networkedrigidbody ();
  cout << endl;

  cout << " UUVBody () defaults printed using different methods:"
       << uuvbody1;
  uuvbody1.print_uuvbody ();

  choice = '*';  // dummy value to print keyboard menu
  break;

case 'I':  case 'i': //-----------------------------------------------

  cout << "Invert test:  uuvbody.test_invert_matrix ()" << endl;

  uuvbody1.test_invert_matrix    ();

  choice = '*';  // dummy value to print keyboard menu
  break;

case 'E':  case 'e': //-----------------------------------------------

  cout << "dEad reckoning test" << endl;

  cout << "uuvbody.dead_reckon_test_with_execution_level ():" << endl;

  uuvbody1.dead_reckon_test_with_execution_level ();

  choice = '*';  // dummy value to print keyboard menu
  break;

case 'T':  case 't': //-----------------------------------------------

  if (TRACE == 0) TRACE = 1;
  else            TRACE = 0;
```

```
      cout << "Trace toggled, now TRACE = " << TRACE << " (";

      if (TRACE == 0) cout << "FALSE)";
      else            cout << "TRUE)";

      break;

    case 'P':  case 'p': //-----------------------------------------------

      cout << "PDU_skip_interval change (from "
           <<  uuvbody1.PDU_skip_interval_value () << ")" << endl;

      cout << "PDUs will only be sent every <value> tenths of a second."
           << endl;
      cout << "Values less than 10 may make applications prone to crash."
           << endl;
      cout << "Enter new value:  ";

      int  new_skip_value;
      cin >> new_skip_value;
      cout << endl;

      uuvbody1.set_PDU_skip_interval (new_skip_value);

      break;

    case 'C':  case 'c': //-----------------------------------------------

      cout << "Close" << endl;

      cout << "DIS_net_close ()" << endl;

      uuvbody1.DIS_net_close ();

      break;

    case 'Q':  case 'q': //-----------------------------------------------

      cout << "Quit" << endl;
      done = TRUE;
      break;

    default:             //-----------------------------------------------

      if ( ! isspace (choice)) cout << " ? unknown option.." << endl;
      break;

    } // end:  switch (choice)

  } while (done == FALSE);

  // normal exit
}
////////////////////////////////////////////////////////////////////////////////
```

C. *AUVglobals.h* Robot Telemetry Variables

```
/////////////////////////////////////////////////////////////////////////////////
/*
 Header file:     AUVglobals.H

 Author:          Don Brutzman

 Revised:         21 October 94

 System:          Irix 5.2

 Compiler:        ANSI C++

 Compilation:     irix> make dynamics

 Description:     Generalized vehicle state variables, customized for
                  NPS AUV II operation with virtual world hydrodynamics model

 Advisors:        Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 References:      Brutzman, Donald P., "Integrated Simulation for Rapid
                  Development of Autonomous Underwater Vehicles,"
                  IEEE Oceanic Engineering Society Autonomous Underwater
                  Vehicle (AUV) Conference 92, Washington DC, June 4-5 1992,
                  pp. 3-10.

                  Brutzman, Donald P., "NPS AUV Integrated Simulator,"
                  Masters thesis, Naval Postgraduate School, Monterey CA,
                  March 1992.

                  Healey, Anthony J. and Lienard, David, "Multivariable
                  Sliding Mode Control for Autonomous Diving and Steering
                  of Unmanned Underwater Vehicles," IEEE Journal of Oceanic
                  Engineering, vol. 18 no. 3, July 1993, pp. 327-339,

                  Lewis, Edward V., editor, _Principles of Naval
                  Architecture volume III_, second revision, The Society of
                  Naval Architects and Marine Engineers, Jersey City
                  New Jersey, 1988, pp. 188-190 and 418-423.

*/
/////////////////////////////////////////////////////////////////////////////////

#ifndef AUVGLOBALS_H
#define AUVGLOBALS_H    // prevent errors if multiple #includes present

/*******************************************************************************/
/* AUV telemetry state vector                                                  */
/*                                                                             */
/* Note these are globals for direct access by any world model.  Refer to      */
/* individual world models for details.  Data hiding within a private object   */
/* is not necessary since all values are transient and superseded by actual    */
/* state when it occurs.  Additionally, half of the state variables are         */
/* provided only by the AUV microprocessor socket, and the other half are       */
/* provided by respective world models.  Thus global variables in this design  */
/* are not vulnerable to corruption and side effects, making data hiding        */
/* unnecessary.                                                                 */
/*******************************************************************************/
```

```
// AUV-provided state variables - - - - - - - - - - - - - - - - - - - - - - - - - -

static double AUV_time            = 0.0; // mission time
static double AUV_delta_rudder    = 0.0; // positive is bow rudder to starboard
static double AUV_delta_planes    = 0.0; // positive is bow planes to starboard
static double AUV_port_rpm        = 0.0; // propellor revolutions per minute
static double AUV_stbd_rpm        = 0.0; // propellor revolutions per minute

static double AUV_bow_vertical    = 0.0; // thruster volts 24V = 3820 rpm no load
static double AUV_stern_vertical  = 0.0; // thruster volts 24V = 3820 rpm no load
static double AUV_bow_lateral     = 0.0; // thruster volts 24V = 3820 rpm no load
static double AUV_stern_lateral   = 0.0; // thruster volts 24V = 3820 rpm no load

static double AUV_depth_cell      = 0.0; // pressure sensor, units are psid
static double AUV_heading         = 0.0; // gyrocompass in degrees
static double AUV_roll_angle      = 0.0; // matches intertial sensor onboard AUV
static double AUV_pitch_angle     = 0.0; // matches intertial sensor onboard AUV
static double AUV_roll_rate       = 0.0; // matches intertial sensor onboard AUV
static double AUV_pitch_rate      = 0.0; // matches intertial sensor onboard AUV
static double AUV_yaw_rate        = 0.0; // matches intertial sensor onboard AUV

static int    AUV_hour            = 0;   // internal AUV OS-9 system time, unused
static int    AUV_minute          = 0;
static int    AUV_second          = 0;


// Hydrodynamics-model-provided state variables - - - - - - - - - - - - - - - - -

static double AUV_x               = 0.0; // x     position in world coordinates
static double AUV_y               = 0.0; // y     position in world coordinates
static double AUV_z               = 0.0; // z     position in world coordinates
static double AUV_phi             = 0.0; // roll  posture in world coordinates
static double AUV_theta           = 0.0; // pitch posture in world coordinates
static double AUV_psi             = 0.0; // yaw   posture in world coordinates

static double AUV_x_dot           = 0.0; //      Euler velocity along North-axis
static double AUV_y_dot           = 0.0; //      Euler velocity along  East-axis
static double AUV_z_dot           = 0.0; //      Euler velocity along Depth-axis
static double AUV_phi_dot         = 0.0; // Euler rotation rate about North-axis
static double AUV_theta_dot       = 0.0; // Euler rotation rate about  East-axis
static double AUV_psi_dot         = 0.0; // Euler rotation rate about Depth-axis

static double AUV_u               = 0.0; // surge  linear velocity along x-axis
static double AUV_v               = 0.0; // sway   linear velocity along y-axis
static double AUV_w               = 0.0; // heave  linear velocity along x-axis
static double AUV_p               = 0.0; // roll  angular velocity about x-axis
static double AUV_q               = 0.0; // pitch angular velocity about y-axis
static double AUV_r               = 0.0; // yaw   angular velocity about z-axis

static double AUV_u_dot           = 0.0; //    linear acceleration along x-axis
static double AUV_v_dot           = 0.0; //    linear acceleration along y-axis
static double AUV_w_dot           = 0.0; //    linear acceleration along x-axis
static double AUV_p_dot           = 0.0; //   angular acceleration about x-axis
static double AUV_q_dot           = 0.0; //   angular acceleration about y-axis
static double AUV_r_dot           = 0.0; //   angular acceleration about z-axis

static double AUV_oceancurrent_u = 0.0; //  Ocean current rate along North-axis
static double AUV_oceancurrent_v = 0.0; //  Ocean current rate along  East-axis
static double AUV_oceancurrent_w = 0.0; //  Ocean current rate along Depth-axis
```

```
// Prior time loop saved variables - - - - - - - - - - - - - - - - - - - - - - - - -

static double AUV_time_prior     = 0.0; // mission time

static double AUV_x_prior        = 0.0; // x    position in world coordinates
static double AUV_y_prior        = 0.0; // y    position in world coordinates
static double AUV_z_prior        = 0.0; // z    position in world coordinates

static double AUV_phi_prior      = 0.0; // roll  posture in world coordinates
static double AUV_theta_prior    = 0.0; // pitch posture in world coordinates
static double AUV_psi_prior      = 0.0; // yaw   posture in world coordinates

// Sonar model provided state variables  - - - - - - - - - - - - - - - - - - - - -

static double AUV_ST1000_bearing = 0.0; // ST_1000 conical pencil beam bearing
static double AUV_ST1000_range   = 0.0; // ST_1000 conical pencil beam range
static double AUV_ST1000_strength= 0.0; // ST_1000 conical pencil beam strength
const  double AUV_ST1000_x_offset= 3.5; // ST_1000 longitudinal location in feet

static double AUV_ST725_bearing  = 0.0; // ST_725  1 x 24  sector beam bearing
static double AUV_ST725_range    = 0.0; // ST_725  1 x 24  sector beam range
static double AUV_ST725_strength = 0.0; // ST_725  1 x 24  sector beam strength
const  double AUV_ST725_x_offset = 3.5; // ST_725  longitudinal location in feet

// Model-wide global variables and constants - - - - - - - - - - - - - - - - - - -

static int    TRACE              = 0;         /* 1 = trace on, 0 = trace off */

#define TRUE  1
#define FALSE 0

#define PI    3.1415926535897932

/*****************************************************************************/

#endif     // endif_H
```

D. *UUVModel.h* Hydrodynamics Model Coefficients

```
////////////////////////////////////////////////////////////////////////////////
/*
 Program:          UUVmodel.H

 Author:           Don Brutzman

 Revised:          26 October 94

 System:           Irix 5.2

 Compiler:         ANSI C++

 Compilation:      irix> make dynamics
                   irix> CC UUVmodel.H +w

                    +w == Warn about all questionable constructs.

 Debugging:        gravy1:~brutzman/dynamics>> lint UUVmodel.h

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 References:       Healey, Anthony J. and Lienard, David, "Multivariable
                   Sliding Mode Control for Autonomous Diving and Steering
                   of Unmanned Underwater Vehicles," IEEE Journal of Oceanic
                   Engineering, vol. 18 no. 3, July 1993, pp. 327-339,

                   Lewis, Edward V., editor, _Principles of Naval
                   Architecture volume III_, second revision, The Society of
                   Naval Architects and Marine Engineers, Jersey City
                   New Jersey, 1988, pp. 188-190 and 418-423.

                   Gertler, Morton and Hagen, Grant R., _Standard Equations
                   of Motion for Submarine Simulation_, Naval Ship
                   Research and Development Center (NSRDC) Research and
                   Development Report 2510, Washington DC, June 1967.

                   Smith, N.S., Crane J.W. and Summey, D.C., _SDV Simulator
                   Hydrodynamic Coefficients_, Naval Coastal Systems Center
                   (NCSC), Panama City Florida, June 1978.  Declassified.

                   Marco, David.  "Slow Speed Control and Dynamic Positioning
                   of an Autonomous Vehicle," Ph.D. dissertation,
                   Naval Postgraduate School, Monterey California, March 1995.

                   Bahrke, Fredric G., "On-Line Identificaton of the Speed,
                   Steering and Diving Response Parameters of an Autonomous
                   Underwater Vehicle from Experimental Data," Master's Thesis,
                   Naval Postgraduate School, Monterey California, March 1992.

                   Warner, David C., "Design, Simulation and Experimental
                   Verification of a Computer Model and Enhanced Position
                   Estimator for the NPS AUV II," Master's Thesis,
                   Naval Postgraduate School, Monterey California, December 1991.

 Notes:            const definitions are for software engineering reliability
                     they can be changed to variables if coefficient modification
                     becomes desirable

*/
////////////////////////////////////////////////////////////////////////////////

#ifndef UUVMODEL_H
#define UUVMODEL_H     // prevent errors if multiple #includes present
```

```
// #define SI          // <<<<<<<<<<<<<<<<  uncomment this statement for SI units
                       //                   otherwise standard British units used

//--------------------------------------------------------------------------//
//            term       value        units      description
//--------------------------------------------------------------------------//

#ifdef SI      // Systeme International (metric) units -----------------------

const double  Weight  = 435.0*0.454; // N          Weight   (0.454 kg/lb == 1)
                   // = 197.49
const double  Buoyancy= 435.0*0.454; // N          Buoyancy (0.454 kg/lb == 1)
                   // = 197.49
const double  L       =    2.23    ; // m          characteristic length 87.625"
const double  g       =    9.81    ; // m/s^2      gravitational constant
const double  rho     = 1000.0     ; // kg/m^3     mass density of fresh water
const double  m       = Weight / g ; // N-s^2/m    vehicle mass incl. free flood
                   // =   20.131498

#define       m4_ft4   (0.305)*(^P.305)*(0.305)*(0.305) // (0.305 m/ft == 1)

                                   ; //            Inertia matrix coefficients
const double  I_x     = 2.7*m4_ft4 ; // Nms^2      = I_xx = 0.023364857
const double  I_y     = 2.7*m4_ft4 ; // Nms^2      = I_yy = 0.023364857
const double  I_z     = 2.7*m4_ft4 ; // Nms^2      = I_zz = 0.023364857
const double  I_xy    =    0.0     ; // Nms^2      = I_yx
const double  I_xz    =    0.0     ; // Nms^2      = I_zx
const double  I_yz    =    0.0     ; // Nms^2      = I_zy


#undef        m4_ft4
                                     //            Centers of Gravity & Buoyancy
const double  x_G     =    0.0     ; // cm
const double  y_G     =    0.0     ; // cm
const double  z_G     =    6.1     ; // cm         Note CG below CB   Marco 0.5"
const double  x_B     =    0.0     ; // cm
const double  y_B     =    0.0     ; // cm
const double  z_B     =    0.0     ; // cm         CB at center of UUV

#else // (not SI) standard British units ------------------------------------

const double  Weight  =  435.0     ; // lb         Weight
const double  Buoyancy=  435.0     ; // lb         Buoyancy
const double  L       = 7.30208333 ; // ft         characteristic length 87.625"
const double  g       =   32.174   ; // ft/s^2     gravitational constant
const double  rho     =    1.94    ; // slugs/ft^3 mass density of fresh water
const double  m       = Weight / g ; // lb/ft-s^2  vehicle mass incl. free flood
                   // =   13.520234

                                   ; //            Inertia matrix coefficients
const double  I_x     =    2.7     ; // lb-ft-sec^2ft^4      =I_xx
const double  I_y     =   42.0     ; // ft^4       =I_yy
const double  I_z     =   45.0     ; // ft^4       =I_zz
const double  I_xy    =    0.0     ; // ft^4       =I_yx
const double  I_xz    =    0.0     ; // ft^4       =I_zx
const double  I_yz    =    0.0     ; // ft^4       =I_zy

                                     //            Centers of Gravity & Buoyancy
const double  x_G     = 0.125/12.0 ; // ft         0.010416667
const double  y_G     =    0.0     ; // ft
const double  z_G     = 1.07 /12.0 ; // ft         Note CG below CB
const double  x_B     = 0.125/12.0 ; // ft         0.010416667
const double  y_B     =    0.0     ; // ft
const double  z_B     =    0.0     ; // ft         CB at center of UUV

// Thruster/propeller distances from centerlines.  Note stern/port are negative.
```

```
const double x_bow_vertical   =   1.41 ; // ft   Marco   17"  measured   13.0"
const double x_stern_vertical = - 1.41 ; // ft   Marco - 17"  measured - 21.0"
const double x_bow_lateral    =   1.92 ; // ft   Marco   23"  measured   19.0"
const double x_stern_lateral  = - 1.92 ; // ft   Marco - 23"  measured - 26.0"

const double y_port_propeller = - 0.313; // ft   Marco - 4"   measured - 3.75"
const double y_stbd_propeller =   0.313; // ft   Marco   4"   measured   3.75"

#endif

//-------------------------------------------------------------------------------//
//              Surge equation of motion coefficients                            //

const double X_u_dot =    -2.82E-3  ; //  Linear force coefficients acting in
const double X_v_dot =     0.0      ; //     the longitudinal body axis
const double X_w_dot =     0.0      ; //     with respect to subscripted
const double X_p_dot =     0.0      ; //           motion components
const double X_q_dot =     0.0      ; //
const double X_r_dot =     0.0      ; //

const double X_uu    =     0.0      ; //
const double X_vv    =     0.0      ; // old -1.743E-2  SDV-9 holdover?
const double X_ww    =     0.0      ; //
const double X_pp    =     0.0      ; //
const double X_qq    =     0.0      ; //
const double X_rr    =     0.0      ; // old -7.53E-3   SDV-9 holdover?

const double X_prop  =     7.78E-3  ; //   X_prop "constant" no longer
applicable

const double X_rb    =     0.283 * L; //
const double X_rs    =    -0.377 * L; //

const double X_uu_delta_b_delta_b = -1.018E-2 ; // drag due to  bow  plane
const double X_uu_delta_s_delta_s = -1.018E-2 ; // drag due to stern plane
const double X_uu_delta_r_delta_r = -1.018E-2 ; // drag due to rudder

const double X_pr    =     0.0      ; //  (these aren't in Bahrke thesis model)
const double X_wq    =     0.0      ; //
const double X_vp    =     0.0      ; //
const double X_vr    =     0.0      ; //

const double X_uq_delta_bow    =   0.0      ; //
const double X_uq_delta_stern  =   0.0      ; //
const double X_ur_delta_rudder =   0.0      ; //
const double X_uv_delta_rudder =   0.0      ; //
const double X_uw_delta_bow    =   0.0      ; //
const double X_uw_delta_stern  =   0.0      ; //

const double X_qdsn            =   0.0      ; //  no longer used in new model
const double X_wdsn            =   0.0      ; //  no longer used in new model
const double X_dsdsn           =   0.0      ; //  no longer used in new model

const double speed_per_rpm = 2.0 / 700.0    ; // steady state:  0.0028571429
                                              // = (2.0 feet/sec) per 700 rpm
const double C_d0    =     0.00778          ; //

//-------------------------------------------------------------------------------//
//              Sway  equation of motion coefficients                            //

const double Y_u_dot =     0.0      ; //  Linear force coefficients acting in
const double Y_v_dot =    -3.43E-2  ; //     the athwartships body axis
const double Y_w_dot =     0.0      ; //     with respect to subscripted
const double Y_p_dot =     0.0      ; //           motion components
const double Y_q_dot =     0.0      ; //
const double Y_r_dot =    -1.78E-3  ; //
```

```
const double  Y_uu    =     0.0       ; //
const double  Y_uv    =    -1.07E-1   ; //
const double  Y_uw    =     0.0       ; //
const double  Y_up    =     0.0       ; //
const double  Y_uq    =     0.0       ; //
const double  Y_ur    =     0.0       ; // Warner = 1.187E-2;  Bahrke = 0.0

const double  Y_uu_delta_rb =  1.18E-2  ; // Marco = 1.241E-2;  Bahrke = 1.18E-2
const double  Y_uu_delta_rs =  1.18E-2  ; // Marco = 1.241E-2;  Bahrke = 1.18E-2

const double  Y_pq    =     0.0       ; //  (these aren't in Bahrke thesis model)
const double  Y_qr    =     0.0       ; //
const double  Y_vq    =     0.0       ; //
const double  Y_wp    =     0.0       ; //
const double  Y_wr    =     0.0       ; //
const double  Y_vw    =     0.0       ; //

const double  C_dy    =     0.5       ; //

//----------------------------------------------------------------------------//
//             Heave equation of motion coefficients                          //

const double  Z_u_dot =     0.0       ; //  Linear force coefficients acting in
const double  Z_v_dot =     0.0       ; //       the vertical body axis
const double  Z_w_dot =    -9.34E-2   ; //     with respect to subscripted
const double  Z_p_dot =     0.0       ; //          motion components
const double  Z_q_dot =    -2.53E-3   ; //
const double  Z_r_dot =     0.0       ; //

const double  Z_vv    =     0.0       ; //
const double  Z_uw    =    -7.844E-1  ; //
const double  Z_up    =     0.0       ; //
const double  Z_uq    =     0.0       ; // Marco = -7.013E-2;  Bahrke = 0.0
const double  Z_rr    =     0.0       ; //
const double  Z_pp    =     0.0       ; //

const double  Z_uu_delta_b = - 2.11E-2  ; //
const double  Z_uu_delta_s = - 2.11E-2  ; //

const double  Z_pr    =     0.0       ; //  (these aren't in Bahrke thesis model)
const double  Z_vp    =     0.0       ; //
const double  Z_vr    =     0.0       ; //

const double  Z_qn    =     0.0       ; //    no longer used in new model
const double  Z_wn    =     0.0       ; //    no longer used in new model
const double  Z_dsn   =     0.0       ; //    no longer used in new model

const double  C_dz    =     0.6       ; //

//----------------------------------------------------------------------------//
//             Roll  equation of motion coefficients                          //

const double  K_u_dot =     0.0       ; //  Angular force coefficient acting
const double  K_v_dot =     0.0       ; //  about the longitudinal body axis
const double  K_w_dot =     0.0       ; //    with respect to subscripted
const double  K_p_dot =    -2.4E-4    ; //          motion components
const double  K_q_dot =     0.0       ; //
const double  K_r_dot =     0.0       ; //

const double  K_uu    =     0.0       ; //
const double  K_uv    =     0.0       ; //
const double  K_uw    =     0.0       ; //
const double  K_up    =    -5.4E-3    ; // surge-related roll damping drag
const double  K_uq    =     0.0       ; //
const double  K_ur    =     0.0       ; //

const double  K_uu_planes = 0.0       ; //  (these aren't in Bahrke thesis model)
```

```
const double  K_pq    =     0.0      ; //
const double  K_qr    =     0.0      ; //
const double  K_vq    =     0.0      ; //
const double  K_wp    =     0.0      ; //
const double  K_wr    =     0.0      ; //
const double  K_vw    =     0.0      ; //
const double  K_prop  =     0.0      ; // K_prop "constant" no longer applicable

const double  K_pn    =     0.0      ; // no longer used in new model

const double  K_pp    =    -2.02E-2  ; // test value for p-squared damping
                                       //   static roll damping drag
                                       // 100 * Healey minimal estimate
                                       //        (-2.02E-4)

const double  K_p     =   K_pp/57.3  ; // estimate based on quadratic term
                                       // (K_pp) equivalent damping at 1 deg/sec


//----------------------------------------------------------------------------//
//              Pitch equation of motion coefficients                         //

const double  M_u_dot =     0.0      ; //  Angular force coefficient acting
const double  M_v_dot =     0.0      ; //  about the athwartships body axis
const double  M_w_dot =    -2.53E-3  ; //    with respect to subscripted
const double  M_p_dot =     0.0      ; //          motion components
const double  M_q_dot =    -6.25E-3  ; //
const double  M_r_dot =     0.0      ; //

const double  M_uu    =     0.0      ; //
const double  M_vv    =     0.0      ; //
const double  M_uw    =     0.0      ; //
const double  M_pp    =     0.0      ; //
const double  M_rr    =     0.0      ; //

const double  M_uq    =    -1.53E-2  ; // surge-related pitch damping drag ***

const double  M_uu_delta_bow    =  - 0.283 * L * Z_uu_delta_b;
                    //               note (-) Z_uu_delta_b
                    //      =    0.043602433

const double  M_uu_delta_stern =  + 0.377 * L * Z_uu_delta_s;
                    //               note (-) Z_uu_delta_s
                    //      =  - 0.058085219

const double  M_pr    =     0.0      ; //  (these aren't in Bahrke thesis model)
const double  M_vp    =     0.0      ; //
const double  M_vr    =     0.0      ; //
const double  M_prop  =     0.0      ; // M_prop "constant" no longer applicable

const double  M_qn    =     0.0      ; // no longer used in new model
const double  M_wn    =     0.0      ; // no longer used in new model
const double  M_dsn   =     0.0      ; // no longer used in new model

const double  M_qq    =    -7.00E-3  ; // slightly larger than N_rr estimate
                                       // test value for q-squared
                                       //   static pitch damping drag
                                       // estimated M_qq ~ K_pp * length / width
                                       // Torsiello  ~ 0.005* 7.3'/ 10.1' = .005

const double  M_q     =  M_qq / 57.3;  // estimate based on quadratic term
                                       // (M_qq) equivalent damping at 1 deg/sec


//----------------------------------------------------------------------------//
//              Yaw   equation of motion coefficients                         //
```

```
const double  N_u_dot =      0.0      ; //  Angular force coefficient acting
const double  N_v_dot =     -1.78E-3  ; //    about the vertical body axis
const double  N_w_dot =      0.0      ; //    with respect to subscripted
const double  N_p_dot =      0.0      ; //          motion components
const double  N_q_dot =      0.0      ; //
const double  N_r_dot =     -4.7E-4   ; //

const double  N_uu    =      0.0      ; //
const double  N_uv    =      0.0      ; //  Marco = -7.69E-3;
const double  N_uw    =      0.0      ; //
const double  N_up    =      0.0      ; //
const double  N_uq    =      0.0      ; //
const double  N_ur    =     -3.90E-3  ; // surge-related yaw damping drag

// N_uu_delta_rb and N_uu_delta_rs not symmetric due to different moment arms

// const double N_uu_delta_rb = 1.3259   ; // Marco
// const double N_uu_delta_rs = 1.7663   ; // Marco

const double  N_uu_delta_rb =  0.283 * L * Y_uu_delta_rb; // Bahrke  =
0.02437762
const double  N_uu_delta_rs =  0.377 * L * Y_uu_delta_rs; // Bahrke  =
0.03247478

const double  N_prop  =      0.0      ; // Normally 0.0 yaw moment due to paired
                                        // counter-rotating propellors;
           // however N_prop is not zero if propellor rpms are independent
           // thus yaw equation of motion now has yaw moments due to propellers
           // and N_prop "constant" is no longer applicable

const double  N_pq    =      0.0      ; //  (these aren't in Bahrke thesis model)
const double  N_qr    =      0.0      ; //
const double  N_vq    =      0.0      ; //
const double  N_wp    =      0.0      ; //
const double  N_wr    =      0.0      ; //
const double  N_vw    =      0.0      ; //

const double  N_rr    =     -5.48E-3  ; // Torsiello value p.113 adjusted for L^5
                                        // correction;  static yaw damping drag
                                        // estimated  N_rr ~ M_qq * height/ width
                                        //                 = .040 * 10.1" / 16.5"
                                        //                 = 0.005
                                        // Torsiello:  0.005
                                        // Healey:     N_rr ~ M_qq

// alternate N_rr = 2 * 2# * 1.92' /(rho/2 L^5 * r_max * r_max) => 0.0048473244
//                  using r_max = 16 deg/sec Torsiello which is consistent

const double  N_r     =  N_rr / 57.3;  // estimate based on quadratic term
                                        // (N_rr) equivalent damping at 1 deg/sec

//-----------------------------------------------------------------------//

// from Dave Marco's dynamics model:
// DEFINE THE LENGTH X, BREADTH BR, AND HEIGHT HH TERMS

   double xx [15] = {
   -43.9/12.0,
   -39.2/12.0,
   -35.2/12.0,
   -31.2/12.0,
   -27.2/12.0,
   -10.0/12.0,
    0.0/12.0,
    10.0/12.0,
    26.8/12.0,
    32.0/12.0,
```

```
  37.8/12.0,
  40.8/12.0,
  42.3/12.0,
  43.3/12.0,
  43.7/12.0};

 double hh [15] = {
 0.0/12.0,
 2.7/12.0,
 5.2/12.0,
 7.6/12.0,
 10.1/12.0,
 10.1/12.0,
 10.1/12.0,
 10.1/12.0,
 10.1/12.0,
 9.6/12.0,
 7.6/12.0,
 5.6/12.0,
 4.2/12.0,
 2.3/12.0,
 0.0/12.0};

 double bb [15] = {
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 16.5/12.0,
 15.5/12.0,
 12.4/12.0,
 9.5/12.0,
 7.0/12.0,
 4.0/12.0,
 0.0/12.0};

//-----------------------------------------------------------------------//

#endif      // UUVMODEL_H
```

E. *UUVBody.C* Unmanned Underwater Vehicle Networked Rigid Body

```
////////////////////////////////////////////////////////////////////////////////
/*
 Program:          UUVBody.C

 Description:      Six degree-of-freedom underwater vehicle hydrodynamics
                   based on Healey model

 Revised:          28 October 94

 System:           Irix 5.2

 Compiler:         ANSI C++

 Compilation:      irix> make dynamics
                   irix> CC UUVBody.C -lm -c -g +w

                   -c == Produce binaries only, suppressing the link phase.
                   +w == Warn about all questionable constructs.

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 Author:           Don Brutzman                  brutzman@cs.nps.navy.mil
                   Code OR/Br
                   Naval Postgraduate School     (408) 656-2149 work
                   Monterey CA 93943-5000        (408) 656-2595 fax

 References:       Healey, A.J. and Lienard, D., "Multivariable Sliding Mode
                   Control for Autonomous DIving and Steering of Unmanned
                   Underwater Vehicles," IEEE Journal of Oceanic Engineering,
                   vol. 18 no. 3, July 1993, pp. 327-339.

                   Yuh, J., "Modeling and Control of Underwater Robotic
                   Vehicle," IEEE Transactions on Systems, Man and Cybernetics,
                   vol. 20 no. 6, November/December 1990, pp. 1475-1483.

                   Press, William H., Teukolsky, Saul A., Vetterling,
                   William T. and Flannery, Brian P., "Numerical Recipes in C,"
                   second edition, Cambridge University Press, Cambridge
                   England, 1992.

                   Marco, David. "Slow Speed Control and Dynamic Positioning
                   of an Autonomous Vehicle," Ph.D. dissertation,
                   Naval Postgraduate School, Monterey California, March 1995.

                   Fossen, Thor I., _Guidance and Control of Ocean Vehicles_,
                   John Wiley and Sons, Chichester England, 1994.

 Status:           Equations of motion tested satisfactorily,
                   verification against in-water tests remains.
                   Move utilities to math_utilities.c

 Future work:      Comments and suggestions are welcome!

*/
////////////////////////////////////////////////////////////////////////////////

#ifndef UUVBODY_C
#define UUVBODY_C    // prevent errors if multiple #includes present

char * DISCLIENT = "~/DIS.mcast/src/client -r &";
```

```
#include "SonarModel.C"

#include "UUVmodel.H"

#include "DISNetworkedRigidBody.C"

////////////////////////////////////////////////////////////////////////////////////

class UUVBody : public DISNetworkedRigidBody
{
private:

Hmatrix Hprevious ;

// member data fields

   double     current_uuv_time;

   double     U;                        // Linear & angular velocities
   double     V;                        //   in body coordinates
   double     W;                        //   (note:  RigidBody is world coordinates)
   double     P;
   double     Q;
   double     R;

   double     u_dot;                    // Linear & angular accelerations
   double     v_dot;                    //   in body coordinates
   double     w_dot;
   double     p_dot;
   double     q_dot;
   double     r_dot;

   double     x_dot;                    // Euler velocities, world coordinates
   double     y_dot;
   double     z_dot;

   double     phi_dot;                  // Euler rotation rates, world coordinates
   double     theta_dot;
   double     psi_dot;

   double     mass [6][6];              // mass matrix acceleration coefficients

   double     mass_inverse [6][6];      // mass matrix inverted

   double     rhs [6];                  // right-hand-sides of equations of motion

   double     new_acceleration [6];     // product [mass_inverse][rhs]

   double     new_velocity [6];         // (averaged_accelerations * dt)
                                        // + old_velocities

   // AUV telemetry state vector is located in "AUV_globals.h" file

public:

// member constructor and destructor functions

   UUVBody                                      ();

  ~UUVBody                                      ()      { /* null body */ }

// operators

   void       multiply6x6_6                     (double   left_matrix   [6][6],
                                                 double   right_matrix  [6],
                                                 double   result_matrix [6]);
```

```
void        multiply6x6_6x6                (double   left_matrix   [6][6],
                                            double   right_matrix  [6][6],
                                            double   result_matrix [6][6]);

friend ostream& operator <<                (ostream& out, UUVBody& uuvb);

// inspection methods

void        print_uuvbody                  ();

void        print_matrix6                  (double output_matrix [6]);
void        print_matrix6x6                (double output_matrix [6][6]);

double      current_uuv_time_value         () const;

double      u_dot_value                    () const;
double      v_dot_value                    () const;
double      w_dot_value                    () const;
double      p_dot_value                    () const;
double      q_dot_value                    () const;
double      r_dot_value                    () const;

double      x_dot_value                    () const;
double      y_dot_value                    () const;
double      z_dot_value                    () const;

// modifying methods

void        UUVBody_initialize             ();

void        set_current_uuv_time           (double new_current_uuv_time);

void        set_accelerations              (double new_u_dot,
                                            double new_v_dot,
                                            double new_w_dot,
                                            double new_p_dot,
                                            double new_q_dot,
                                            double new_r_dot);

void        set_linear_accelerations       (double new_u_dot,
                                            double new_v_dot,
                                            double new_w_dot);

void        set_angular_accelerations      (double new_p_dot,
                                            double new_q_dot,
                                            double new_r_dot);

          // vehicle socket communications with dynamics
void        loop_test_with_execution_level ();

          // vehicle socket communications with no dynamics
void        dead_reckon_test_with_execution_level ();

void        swap                           (double * a, double * b);

double      square                         (double value);

double      nonzerosign                    (double value);

void        clamp                          (double * clampee,
                                            double   absolute_min,
                                            double   absolute_max,
                                            char   * name);

double      epsilon                        ();

void        invert_mass_matrix             ();
```

```
   void       test_invert_matrix            ();

   void       calculate_mass_matrix         ();

   void       integrate_equations_of_motion ();

};

/////////////////////////////////////////////////////////////////////////////////

//----------------------------------------------------------------------------//
// constructor methods
//----------------------------------------------------------------------------//

UUVBody:: UUVBody () // default constructor
{
   Hprevious = Hmatrix ();

   RigidBody_initialize ();                      // inherited method
   DISNetworkedRigidBody_initialize ();          // inherited method

   current_uuv_time = 0.0;

   U               = 0.0;
   V               = 0.0;
   W               = 0.0;
   P               = 0.0;
   Q               = 0.0;
   R               = 0.0;

   u_dot           = 0.0;
   v_dot           = 0.0;
   w_dot           = 0.0;
   p_dot           = 0.0;
   q_dot           = 0.0;
   r_dot           = 0.0;

   x_dot           = 0.0;
   y_dot           = 0.0;
   z_dot           = 0.0;
   phi_dot         = 0.0;
   theta_dot       = 0.0;
   psi_dot         = 0.0;

   for (int index = 0; index <= 5; index++) rhs [index] = 0.0;

   calculate_mass_matrix ();
   invert_mass_matrix    ();

}
//----------------------------------------------------------------------------//
// operators
//----------------------------------------------------------------------------//

ostream& operator << (ostream& out, UUVBody& uuvb)
{
   int row, col;

   uuvb.print_networkedrigidbody ();
   out  << "current_uuv_time = "  << uuvb.current_uuv_time << endl;
   out  << "<u_dot, v_dot, w_dot, p_dot, q_dot, r_dot> = " << "<"
        << uuvb.u_dot << ", " << uuvb.v_dot << ", " << uuvb.w_dot << ", "
        << uuvb.p_dot << ", " << uuvb.q_dot << ", " << uuvb.r_dot << ">"
        << endl;

   out  << "Mass matrix:" << endl;
```

```
    for (row = 0; row <= 5; row ++)
    {
         out << "[";
         for (col = 0; col <= 5; col ++)
         {
              out << uuvb.mass [row][col];
              if   (col < 5) out << ", ";
              else           out << "]" << endl;
         }
    }
    out  << "Mass inverse matrix:" << endl;
    for (row = 0; row <= 5; row ++)
    {
         out << "[";
         for (col = 0; col <= 5; col ++)
         {
              out << uuvb.mass_inverse [row][col];
              if   (col < 5) out << ", ";
              else           out << "]" << endl;
         }
    }
    return (out);
}
//-------------------------------------------------------------------------------//

void UUVBody:: multiply6x6_6 (double   left_matrix   [6][6],
                              double   right_matrix  [6],
                              double   result_matrix [6])
{
    int row, col;

    for (row = 0; row <= 5; row++)
    {
        result_matrix [row] = 0.0;

        for (col = 0; col <= 5; col++)
        {
                result_matrix [row] = result_matrix [row] +
                                 left_matrix [row][col] * right_matrix [row];
        }
    }
}
//-------------------------------------------------------------------------------//

void UUVBody:: multiply6x6_6x6 (double   left_matrix   [6][6],
                                double   right_matrix  [6][6],
                                double   result_matrix [6][6])
{
    int row, col, index;

    for (row = 0; row <= 5; row++)
    {
        for (col = 0; col <= 5; col++)
        {
            result_matrix [row][col] = 0.0;

            for (index = 0; index <= 5; index++)
            {
                result_matrix [row][col] = result_matrix [row][col] +

                           left_matrix [row][index] * right_matrix [index][col];
            }
        }
    }
}
//-------------------------------------------------------------------------------//
// inspection methods
```

```
//------------------------------------------------------------------------------//

void UUVBody:: print_uuvbody ()
{
   print_networkedrigidbody ();
   cout << "current_uuv_time   = "  << current_uuv_time << endl;

   cout << "<U, U, W, P, Q, R> body = " << "<"
        << U << ", " << V << ", " << W << ", "
        << P << ", " << Q << ", " << R << ">" << endl;

   cout << "<u_dot, v_dot, w_dot, p_dot, q_dot, r_dot> body = " << "<"
        << u_dot << ", " << v_dot << ", " << w_dot << ", "
        << p_dot << ", " << q_dot << ", " << r_dot << ">" << endl;

   Vector3D euler_position_rates =  Vector3D (x_dot, y_dot, z_dot);
   cout  << "<x_dot,    y_dot,     z_dot>    = " << euler_position_rates << endl;

   Vector3D euler_angular_rates =  Vector3D (phi_dot, theta_dot, psi_dot);
   cout  << "<phi_dot, theta_dot, psi_dot> = " << euler_angular_rates  << endl;

   cout  << "Mass matrix:" << endl;
   print_matrix6x6 (mass);

   cout  << "Mass inverse matrix:" << endl;
   print_matrix6x6 (mass_inverse);

   cout  << "Right-hand side (RHS) of equations of motion:  ";
   print_matrix6 (rhs);
}
//------------------------------------------------------------------------------//

void UUVBody:: print_matrix6 (double output_matrix [6])
{
   int row;

   cout << "[";
   for (row = 0; row <= 5; row ++)
   {
       cout << output_matrix [row];
       if (row < 5) cout << ", ";
   }
   cout << "]" << endl;
}
//------------------------------------------------------------------------------//

void UUVBody:: print_matrix6x6 (double output_matrix [6][6])
{
   int row, col;

   for (row = 0; row <= 5; row ++)
   {
        cout << "[";
        for (col = 0; col <= 5; col ++)
        {
             cout << output_matrix [row][col];
             if   (col < 5) cout << ", ";
             else           cout << "]" << endl;
        }
   }
}
//------------------------------------------------------------------------------//

double UUVBody:: current_uuv_time_value ()
const
{
   return current_uuv_time;
```

```
}
//----------------------------------------------------------------------//

double UUVBody:: u_dot_value ()
const
{
   return u_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: v_dot_value ()
const
{
   return v_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: w_dot_value ()
const
{
   return w_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: p_dot_value ()
const
{
   return p_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: q_dot_value ()
const
{
   return q_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: r_dot_value ()
const
{
   return r_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: x_dot_value ()
const
{
   return x_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: y_dot_value ()
const
{
   return y_dot;
}
//----------------------------------------------------------------------//

double UUVBody:: z_dot_value ()
const
{
   return z_dot;
}
//----------------------------------------------------------------------//
// modifying methods
//----------------------------------------------------------------------//
```

```
void UUVBody:: UUVBody_initialize ()
{
   RigidBody_initialize ();                          // inherited method
   DISNetworkedRigidBody_initialize ();              // inherited method

   current_uuv_time = 0.0;

   U                 = 0.0;
   V                 = 0.0;
   W                 = 0.0;
   P                 = 0.0;
   Q                 = 0.0;
   R                 = 0.0;

   u_dot             = 0.0;
   v_dot             = 0.0;
   w_dot             = 0.0;
   p_dot             = 0.0;
   q_dot             = 0.0;
   r_dot             = 0.0;

   x_dot             = 0.0;
   y_dot             = 0.0;
   z_dot             = 0.0;
   phi_dot           = 0.0;
   theta_dot         = 0.0;
   psi_dot           = 0.0;

   for (int index = 0; index <= 5; index++) rhs [index] = 0.0;

   calculate_mass_matrix ();
   invert_mass_matrix    ();

   AUV_ST1000_bearing = 0.0; // ST_1000 conical pencil beam bearing
   AUV_ST1000_range   = 0.0; // ST_1000 conical pencil beam range
   AUV_ST1000_strength= 0.0; // ST_1000 conical pencil beam strength

   AUV_ST725_bearing  = 0.0; // ST_725  1 x 24  sector beam bearing
   AUV_ST725_range    = 0.0; // ST_725  1 x 24  sector beam range
   AUV_ST725_strength = 0.0; // ST_725  1 x 24  sector beam strength

   AUV_bow_vertical   = 0.0; // thruster volts 24V = 3820 rpm no load
   AUV_stern_vertical = 0.0; // thruster volts 24V = 3820 rpm no load
   AUV_bow_lateral    = 0.0; // thruster volts 24V = 3820 rpm no load
   AUV_stern_lateral  = 0.0; // thruster volts 24V = 3820 rpm no load

   AUV_delta_rudder   = 0.0; // positive is bow rudder to starboard
   AUV_delta_planes   = 0.0; // positive is bow planes to starboard
   AUV_port_rpm       = 0.0; // propellor revolutions per minute
   AUV_stbd_rpm       = 0.0; // propellor revolutions per minute

}
//------------------------------------------------------------------------//

void UUVBody:: set_current_uuv_time (double new_current_uuv_time)
{
   current_uuv_time = new_current_uuv_time;
}
//------------------------------------------------------------------------//

void UUVBody:: set_accelerations          (double new_u_dot, double new_v_dot,
                                            double new_w_dot, double new_p_dot,
                                            double new_q_dot, double new_r_dot)
{
   u_dot = new_u_dot;
   v_dot = new_v_dot;
   w_dot = new_w_dot;
```

```
   p_dot = new_p_dot;
   q_dot = new_q_dot;
   r_dot = new_r_dot;
}
//-----------------------------------------------------------------------//

void UUVBody:: set_linear_accelerations  (double new_u_dot, double new_v_dot,
                                          double new_w_dot)
{
   u_dot = new_u_dot;
   v_dot = new_v_dot;
   w_dot = new_w_dot;
}
//-----------------------------------------------------------------------//

void UUVBody:: set_angular_accelerations (double new_p_dot, double new_q_dot,
                                          double new_r_dot)
{
   p_dot = new_p_dot;
   q_dot = new_q_dot;
   r_dot = new_r_dot;
}
//-----------------------------------------------------------------------//
// vehicle socket communications tests
//-----------------------------------------------------------------------//

void UUVBody:: loop_test_with_execution_level ()
{
   int read_from_socket_result = 0;
   int write_to_socket_result  = 0;

   // print_uuvbody ();  // diagnostic

   if (TRACE) cout << "[loop_test_with_execution_level start]" << endl;

   open_execution_level_socket ();       // repeated calls should not reopen it

   cout << endl;
   cout << "To listen on default multicast port (on another machine):  "
        << endl;
   cout << DISCLIENT << endl;
   cout << endl;

   cout << "DIS_net_open ():" << endl;
   DIS_net_open ();

   while (TRUE)  // loop until break
   {
      read_from_socket_result = read_from_execution_level_socket ();

      if (read_from_socket_result == -4)   // time/position/orientation received
      {
              cout << "position or orientation command received:  <"
                   << AUV_time << "> <"
                   << AUV_x     << ", " << AUV_y     << ", " << AUV_z   << ">  <"
                   << AUV_phi   << ", " << AUV_theta << ", " << AUV_psi << ">"
                   << endl;

              RigidBody_initialize ();

              Vector3D vector1 = Vector3D (AUV_x, AUV_y, AUV_z);

              hmatrix = Hmatrix (vector1,
                                 radians (AUV_phi),
                                 radians (AUV_theta),
                                 radians (AUV_psi));
              hmatrix.print_hmatrix ();
```

```
            Hprevious = hmatrix;
        }
        else if (read_from_socket_result == TRUE) // valid report received
        {
           integrate_equations_of_motion    ();

           if (DIS_net_write () == FALSE)  break; // send out PDU, otherwise done

           test_tank_sonar_model ();  // parallelize & generalize the sonar model

           write_to_execution_level_socket  ();      // send back full telemetry
        }

        if ((read_from_socket_result == -3) || //   KILL   signal was received
            (read_from_socket_result == -4)   ) // position signal was received
        {
            cout << "KILL/position/orientation signal received, " << endl;
            cout << "freeze the AUV where it is." << endl;

            AUV_u          = 0.0;          // surge  linear velocity along x-axis
            AUV_v          = 0.0;          // sway   linear velocity along y-axis
            AUV_w          = 0.0;          // heave  linear velocity along x-axis
            AUV_p          = 0.0;          // roll  angular velocity about x-axis
            AUV_q          = 0.0;          // pitch angular velocity about y-axis
            AUV_r          = 0.0;          // yaw   angular velocity about z-axis

            AUV_u_dot      = 0.0;          //    linear acceleration along x-axis
            AUV_v_dot      = 0.0;          //    linear acceleration along y-axis
            AUV_w_dot      = 0.0;          //    linear acceleration along x-axis
            AUV_p_dot      = 0.0;          //   angular acceleration about x-axis
            AUV_q_dot      = 0.0;          //   angular acceleration about y-axis
            AUV_r_dot      = 0.0;          //   angular acceleration about z-axis

            AUV_x_dot      = 0.0;          //      Euler velocity along North-axis
            AUV_y_dot      = 0.0;          //      Euler velocity along  East-axis
            AUV_z_dot      = 0.0;          //      Euler velocity along Depth-axis
            AUV_phi_dot    = 0.0;          // Euler rotation rate about North-axis
            AUV_theta_dot  = 0.0;          // Euler rotation rate about  East-axis
            AUV_psi_dot    = 0.0;          // Euler rotation rate about Depth-axis

            clock_t start_busywait_clock = clock ();
            while  (clock () < start_busywait_clock + 1.0 * CLOCKS_PER_SEC)
            {
               /* busy wait */
            }
            write_to_socket_result = DIS_net_write ();
            cout << "freeze AUV PDU sent:  " << write_to_socket_result << endl;
        }

        if (read_from_socket_result == -3)   // KILL signal was received
        {
            cout << "KILL signal received, exit "
                 << "UUVBody.loop_test_with_execution_level" << endl;

            if (TRACE) cout << "[loop_test_with_execution_level complete]"
                            << endl;

            return; // all done
        }
    }
    // unreachable exit point
    // shutdown_socket () must have been invoked already to exit preceding loop

} // loop_test_with_execution_level complete
//-------------------------------------------------------------------------//

void UUVBody:: dead_reckon_test_with_execution_level ()
```

```
{
   int read_from_socket_result = 0;
   int write_to_socket_result  = 0;

   cout << endl;
   cout << "To listen on default multicast port:  " << endl;
   cout << DISCLIENT << endl;
   cout << endl;

   open_execution_level_socket ();

   cout << "DIS_net_open ()" << endl;
   DIS_net_open ();

   while (TRUE)   // loop until break
   {
      read_from_socket_result = read_from_execution_level_socket ();

      if ((read_from_socket_result == -2) || // shutdown signal was received
          (read_from_socket_result == -3))   //   quit   signal was received
      {
         cout << "shutdown signal received, freeze the AUV where it is"
              << endl;

         AUV_u          = 0.0;          // surge  linear velocity along x-axis
         AUV_v          = 0.0;          // sway   linear velocity along y-axis
         AUV_w          = 0.0;          // heave  linear velocity along x-axis
         AUV_p          = 0.0;          // roll  angular velocity about x-axis
         AUV_q          = 0.0;          // pitch angular velocity about y-axis
         AUV_r          = 0.0;          // yaw   angular velocity about z-axis

         AUV_u_dot      = 0.0;          //     linear acceleration along x-axis
         AUV_v_dot      = 0.0;          //     linear acceleration along y-axis
         AUV_w_dot      = 0.0;          //     linear acceleration along x-axis
         AUV_p_dot      = 0.0;          //    angular acceleration about x-axis
         AUV_q_dot      = 0.0;          //    angular acceleration about y-axis
         AUV_r_dot      = 0.0;          //    angular acceleration about z-axis

         AUV_x_dot      = 0.0;          //       Euler velocity along North-axis
         AUV_y_dot      = 0.0;          //       Euler velocity along  East-axis
         AUV_z_dot      = 0.0;          //       Euler velocity along Depth-axis
         AUV_phi_dot    = 0.0;          // Euler rotation rate about North-axis
         AUV_theta_dot  = 0.0;          // Euler rotation rate about  East-axis
         AUV_psi_dot    = 0.0;          // Euler rotation rate about Depth-axis

         clock_t start_busywait_clock = clock ();
         while  (clock () < start_busywait_clock + 1.0 * CLOCKS_PER_SEC)
         {
            /* busy wait */
         }
         write_to_socket_result = DIS_net_write ();
         cout << "freeze AUV PDU sent:  " << write_to_socket_result << endl;

         if (read_from_socket_result == -3) // quit signal was received
         {
            cout << "quit signal received, freeze the AUV where it is" << endl;
            cout << "  and rezero for another loop if necessary" << endl;

            UUVBody_initialize ();
         }
         break;
      }

      // equations of motion not integrated, execution level does dead reckoning

      if (DIS_net_write () == FALSE)   break;  // send out a PDU, otherwise done
```

```
      write_to_execution_level_socket  ();      // note state vector preserved
   }
   // shutdown_socket () must have been invoked already to exit preceding loop

   // cout << "DIS_net_close ()" << endl;
   // DIS_net_close ();
}
//--------------------------------------------------------------------------//
// vehicle dynamics functions
//--------------------------------------------------------------------------//

void UUVBody:: swap (double * a, double * b)
{
   double temp = * a;
         * a    = * b;
         * b    = temp;
}
//--------------------------------------------------------------------------//

double UUVBody:: square (double value)
{
   return (value * value);
}
//--------------------------------------------------------------------------//

double UUVBody:: nonzerosign (double value)
{
   if (value != 0.0) return  sign (value);
   else              return      + 1.0;
}
//--------------------------------------------------------------------------//

void UUVBody:: clamp (double * clampee, double absolute_min,
                                        double absolute_max, char * name)
{
   double new_value, local_min, local_max;

   if ((absolute_max == 0.0) && (absolute_min == 0.0)) return;  // no clamp

   if (absolute_max >= absolute_min) // ensure min & max used in proper order
   {
       local_min = absolute_min;
       local_max = absolute_max;
   }
   else
   {
       local_min = absolute_max;
       local_max = absolute_min;
   }
   if ((* clampee) > local_max)
   {
       new_value = local_max;

       cout << "clamping " << name << " from "
            << * clampee << " to " << new_value << endl;

       * clampee = new_value;
   }
   if ((* clampee) < local_min)
   {
       new_value = local_min;

       cout << "clamping " << name << " from "
            << * clampee << " to " << new_value << endl;

       * clampee = new_value;
   }
```

```
}
//------------------------------------------------------------------------//

double UUVBody:: epsilon ()  // not used in revised surge EOM  <<<<<<<<<<<<<<<<<
                             // due to problems in analytic derivation
                             // retained for archival purposes
{
   double        average_rpm = (AUV_port_rpm + AUV_stbd_rpm) / 2.0;

   double n     = average_rpm;

   double eta;

   eta = speed_per_rpm * n / nonzerosign ( U );

// cout << "eta     = " << eta << endl;

   double C_t  = 0.008 * L*L * eta * fabs (eta) / 2.0;

   double C_t1 = 0.008 * L*L                     / 2.0;

   double result;

// X_prop redefinition: net pushing force on vehicle, account for water flow
// X_prop = C_d0 * (eta * fabs (eta) - 1.0);
// X_prop = C_d0 * (eta * fabs (eta));
// cout << "X_prop = " << X_prop << endl;

   result =  -1.0 + (sign (n) / nonzerosign ( U ))

                   * (sqrt (C_t + 1.0) - 1.0) / (sqrt (C_t1 + 1.0) - 1.0);

// kill this function due to rewritten X_propulsion terms
result = 0.0;

// cout << "epsilon () = " << result << endl;

   return result;
}
//------------------------------------------------------------------------//

void UUVBody:: invert_mass_matrix ()    // adapted from Numerical Recipes in C
                                        // 2nd edition pp. 39-40
{
   int         indxc [6], indxr [6], ipiv [6];
   int         i, j, k, row, col, l, ll;
   double      big, dummy, pivinv;
   double      mass_copy [6][6];

   for (i = 0; i <= 5; i++)
   {
       for (j = 0; j <= 5; j++)
       {
           mass_copy [i][j] = mass [i][j];
           if    (i == j) mass_inverse [i][j] = 1.0;
           else           mass_inverse [i][j] = 0.0;
       }
   }
   // int arrays ipiv, indxr and indxc are used for bookkeeping on the pivoting

   for (j = 0; j <= 5; j++) ipiv [j] = 0;   // initialize loop

   for (i = 0; i <= 5; i++)                 // main loop over columns reduced
   {
       big = 0.0;
       for (j = 0; j <= 5; j++)             // outer loop of pivot search
       {
```

```
        if (ipiv [j] != 1)
        {
            for (k = 0; k <= 5; k++)
            {
                if (ipiv [k] == 0)
                {
                    if (fabs (mass_copy [j][k]) >= big)
                    {
                        big = fabs (mass_copy [j][k]);
                        row = j;
                        col = k;
                    }
                }
                else if (ipiv [k] > 1)
                         cout << "Error:  singular mass matrix" << endl;
            }
        }
    }
    ++ (ipiv [col]);
    // see detailed comments in reference code for pivot bookkeeping scheme

    if (row != col)
    {
        for (l = 0; l <= 5; l++) swap (& mass_copy    [row][l],
                                       & mass_copy    [col][l]);
        for (l = 0; l <= 5; l++) swap (& mass_inverse [row][l],
                                       & mass_inverse [col][l]);
    }
    indxr [i] = row; // we are now ready to divide pivot row by pivot element
    indxc [i] = col; //    which is located at [row, col]

    if (mass_copy [col][col] == 0.0)
        cout << "invert error, singular matrix" << endl;

    pivinv = 1.0 / mass_copy [col][col];
    mass_copy [col][col] = 1.0;

    for (l  = 0; l  <= 5;  l++)  mass_copy    [col][l] *= pivinv;
    for (l  = 0; l  <= 5;  l++)  mass_inverse [col][l] *= pivinv;

    for (ll = 0; ll <= 5; ll++)  // next we reduce the rows
    {
         if (ll != col)          // except for the pivot row, of course
         {
             dummy = mass_copy [ll][col];
             mass_copy [ll][col] = 0.0;
             for (l = 0; l <= 5; l++)
                     mass_copy    [ll][l] -= mass_copy    [col][l] * dummy;

             for (l = 0; l <= 5; l++)
                     mass_inverse [ll][l] -= mass_inverse [col][l] * dummy;
         }
    }
  }
  // we now unscramble the columns by interchanging in reverse order

  for (l = 5; l >= 0; l--)
  {
      if (indxr [l] != indxc [l])
      {
          for (k = 0; k <= 5; k++)
          {
             swap (& mass_copy [k][indxr [l]], & mass_copy [k][indxc [l]]);
          }
      }
  }
}
```

```
//------------------------------------------------------------------------//

void UUVBody:: test_invert_matrix ()
{
   int i, j;

   calculate_mass_matrix ();

   cout << "Original mass matrix:" << endl;
   print_matrix6x6 (mass);

   invert_mass_matrix    ();

   cout << endl;
   cout << "Inverted mass matrix:" << endl;
   print_matrix6x6 (mass_inverse);

   for (i = 0; i <= 5; i++)
   {
       for (j = 0; j <= 5; j++)
       {
           mass [i][j] = mass_inverse [i][j];
       }
   }
   invert_mass_matrix    ();

   cout << endl;
   cout << "Double invert_mass_matrix () should get back to "
        << "original mass matrix:" << endl;

   print_matrix6x6 (mass_inverse);

   calculate_mass_matrix ();  // restore
   invert_mass_matrix    ();
}
//------------------------------------------------------------------------//

void UUVBody:: calculate_mass_matrix ()
{
#define UDOT    0           // matrix indices
#define VDOT    1
#define WDOT    2
#define PDOT    3
#define QDOT    4
#define RDOT    5
#define SURGE   0
#define SWAY    1
#define HEAVE   2
#define ROLL    3
#define PITCH   4
#define YAW     5

    mass [SURGE][UDOT] =  m             - 0.5 * rho * L*L*L       * X_u_dot;
    mass [SURGE][VDOT] =  0.0;
    mass [SURGE][WDOT] =  0.0;
    mass [SURGE][PDOT] =  0.0;
    mass [SURGE][QDOT] =  m * ( z_G);
    mass [SURGE][RDOT] =  m * (-y_G);

    mass [SWAY ][UDOT] =  0.0;
    mass [SWAY ][VDOT] =  m             - 0.5 * rho * L*L*L       * Y_v_dot;
    mass [SWAY ][WDOT] =  0.0;
    mass [SWAY ][PDOT] =  m * (-z_G) - 0.5 * rho * L*L*L*L     * Y_p_dot;
    mass [SWAY ][QDOT] =  0.0;
    mass [SWAY ][RDOT] =  m * ( x_G) - 0.5 * rho * L*L*L*L     * Y_r_dot;

    mass [HEAVE][UDOT] =  0.0;
```

```
    mass [HEAVE][VDOT] =  0.0;
    mass [HEAVE][WDOT] =  m           - 0.5 * rho * L*L*L       * Z_w_dot;
    mass [HEAVE][PDOT] =  m * ( y_G);
    mass [HEAVE][QDOT] =  m * (-x_G)  - 0.5 * rho * L*L*L*L     * Z_q_dot;
    mass [HEAVE][RDOT] =  0.0;

    mass [ROLL ][UDOT] =  0.0;
    mass [ROLL ][VDOT] =  m * (-z_G)  - 0.5 * rho * L*L*L*L     * K_v_dot;
    mass [ROLL ][WDOT] =  m * ( y_G);
    mass [ROLL ][PDOT] =  I_x         - 0.5 * rho * L*L*L*L*L   * K_p_dot;
    mass [ROLL ][QDOT] = -I_xy;
    mass [ROLL ][RDOT] = -I_xz        - 0.5 * rho * L*L*L*L*L   * K_r_dot;

    mass [PITCH][UDOT] =  m * ( z_G);
    mass [PITCH][VDOT] =  0.0;
    mass [PITCH][WDOT] =  m * (-x_G)  - 0.5 * rho * L*L*L*L     * M_w_dot;
    mass [PITCH][PDOT] = -I_xy;
    mass [PITCH][QDOT] =  I_y         - 0.5 * rho * L*L*L*L*L   * M_q_dot;
    mass [PITCH][RDOT] = -I_yz;

    mass [YAW  ][UDOT] =  m * (-y_G);
    mass [YAW  ][VDOT] =  m * ( x_G)  - 0.5 * rho * L*L*L*L     * N_v_dot;
    mass [YAW  ][WDOT] =  0.0;
    mass [YAW  ][PDOT] = -I_xz        - 0.5 * rho * L*L*L*L*L   * N_p_dot;
    mass [YAW  ][QDOT] = -I_yz;
    mass [YAW  ][RDOT] =  I_z         - 0.5 * rho * L*L*L*L*L   * N_r_dot;
}

//-------------------------------------------------------------------------//

void UUVBody:: integrate_equations_of_motion ()
{
   current_uuv_time = AUV_time;

   double dt         = current_uuv_time - time_of_posture_value ();

   if (dt < 0.0)     // mission clock was reset, rezero the dynamics model
   {
       current_uuv_time    = AUV_time;
       set_time_of_posture (AUV_time);
       set_velocities     (0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
       set_accelerations  (0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
       dt = 0.0;
       U  = 0.0;
       V  = 0.0;
       W  = 0.0;
       P  = 0.0;
       Q  = 0.0;
       R  = 0.0;
   }

   double rho2         = rho / 2.0;
   double L2           = L * L;
   double L3           = L * L * L;
   double L4           = L * L * L * L;
   double L5           = L * L * L * L * L;

   // note that sign is not preserved in the following squared variables
   //      in order to present consistent naming with Healey reference paper.
   //      To preserve sign, use (U * fabs (U)) etc.
   double P2           = P * P;
   double Q2           = Q * Q;
   double R2           = R * R;
   double U2           = U * U;
   double V2           = V * V;
   double W2           = W * W;
```

```
   // calculate world coordinate posture rates, use holding variables for speed

   double PHI        = phi_value   ();
   double THETA      = theta_value ();
   double PSI        = psi_value   ();

   double sinPHI     = sin (   PHI );
   double cosPHI     = cos (   PHI );
   double sinTHETA   = sin ( THETA );
   double cosTHETA   = cos ( THETA );
   double sinPSI     = sin (   PSI );
   double cosPSI     = cos (   PSI );

// double EPSILON    = epsilon (); // no longer used in revised model

   double delta_planes_stern =   AUV_delta_planes;
   double delta_planes_bow   = - AUV_delta_planes;

   double delta_rudder_stern =   AUV_delta_rudder;
   double delta_rudder_bow   = - AUV_delta_rudder;

   // integrate drag forces over the vehicle - - - - - - - - - - - - - - - - - -//

   double  sway_integral = 0.0;
   double heave_integral = 0.0;
   double pitch_integral = 0.0;
   double   yaw_integral = 0.0;

   double dx, U_cf_x;

   for (int x_index = 0; x_index < 14; x_index ++) // longitudinal centerline
   {
        dx = fabs (xx [x_index] - xx [x_index + 1]);

        U_cf_x = sqrt (  square (V + xx [x_index] * R)

                       + square (W - xx [x_index] * Q));

        if (U_cf_x > 1.0E-6) // arbitrary small non-0 minimum
        {
            sway_integral  +=   rho2 * ( C_dy * hh [x_index]

                                              * square ((V + xx [x_index] * R))

                                       + C_dz * bb [x_index]

                                              * square ((W - xx [x_index] * Q)))

                                     * (V + xx [x_index] * R) * dx / U_cf_x;


            heave_integral +=   rho2 * ( C_dy * hh [x_index]

                                              * square ((V + xx [x_index] * R))

                                       + C_dz * bb [x_index]

                                              * square ((W - xx [x_index] * Q)))

                                     * (W - xx [x_index] * Q) * dx / U_cf_x;


            pitch_integral +=   rho2 * ( C_dy * hh [x_index]

                                              * square ((V + xx [x_index] * R))

                                       + C_dz * bb [x_index]
```

```
                                                    * square ((W - xx [x_index] * Q)))
                                          * (W - xx [x_index] * Q)
                                            // ^ note sign correction
                                          * xx [x_index] * dx / U_cf_x;

            yaw_integral +=    rho2 * ( C_dy * hh [x_index]
                                                    * square ((V + xx [x_index] * R))
                                            + C_dz * bb [x_index]
                                                    * square ((W - xx [x_index] * Q)))
                                          * (V + xx [x_index] * R)
                                          * xx [x_index] * dx / U_cf_x;
        }
    }
    if (TRACE)
    {
       cout << "dx = " << dx << ", U_cf_x = " << U_cf_x
            << ", sway_integral  = " << sway_integral << endl;

       cout << "dx = " << dx << ", U_cf_x = " << U_cf_x
            << ", heave_integral = " << heave_integral << endl;

       cout << "dx = " << dx << ", U_cf_x = " << U_cf_x
            << ", pitch_integral = " << pitch_integral << endl;

       cout << "dx = " << dx << ", U_cf_x = " << U_cf_x
            << ", yaw_integral   = " << yaw_integral << endl;
    }

///////////////////////////////////////////////////////////////////////////////////
// calculate Equations of Motion right-hand sides                                //
///////////////////////////////////////////////////////////////////////////////////

    rhs [SURGE] = // Surge Motion Equation right hand side --------------------//

          m * ((V * R) - (W * Q) + x_G * (Q2 + R2) - y_G * P*Q - z_G * P*R)

        + rho2 * L4 * (  X_pp * P2  + X_qq * Q2
                       + X_rr * R2  + X_pr * P*R)

        + rho2 * L3 * (  X_wq * W*Q + X_vp * V*P + X_vr * V*R

                       + U*Q * (  X_uq_delta_bow     * delta_planes_bow
                                + X_uq_delta_stern   * delta_planes_stern)

                       + U*R * (  X_ur_delta_rudder  * delta_rudder_bow
                                + X_ur_delta_rudder  * delta_rudder_stern)
                      )

        + rho2 * L2 * (  X_vv * V2  + X_ww * W2

                       + U*V * (  X_uv_delta_rudder  * delta_rudder_stern)

                       + U*W * (  X_uw_delta_bow     * delta_planes_bow
                                + X_uw_delta_stern   * delta_planes_stern)

                       + U * fabs (U)  * (  X_uu_delta_b_delta_b
                                          * delta_planes_bow
```

```
                                                    * delta_planes_bow

                                                  + X_uu_delta_s_delta_s
                                                    * delta_planes_stern
                                                    * delta_planes_stern

                                                  + X_uu_delta_r_delta_r
                                                    * delta_rudder_bow
                                                    * delta_rudder_bow

                                                  + X_uu_delta_r_delta_r
                                                    * delta_rudder_stern
                                                    * delta_rudder_stern)
                                 )

        - (Weight - Buoyancy) * sinTHETA

        // EPSILON terms have been removed due to revised equations of motion

        // + rho2 * L3 *    X_qdsn   * U*Q * delta_planes_stern * EPSILON
        // + rho2 * L2 * EPSILON * ( X_wdsn   * U*W * delta_planes_stern
        //
        //                              + X_dsdsn * U2       * delta_planes_stern
        //                                                   * delta_planes_stern)

        // X_propulsion surge force (derived using expressions in Healey paper)
        // note that speed_per_rpm is associated with work of two propellors

        + rho2 * L2 * C_d0 *  square (speed_per_rpm)

                     *  0.5 * (  AUV_port_rpm * fabs (AUV_port_rpm)

                                + AUV_stbd_rpm * fabs (AUV_stbd_rpm))

        // X_resistance surge drag (derived using expressions in Healey paper)

        - rho2 * L2 * C_d0 * U * fabs (U);

///////////////////////////////////////////////////////////////////////////////

if (TRACE)    // Surge TRACE
{
cout << "* surge term1=" <<  m * ((V * R) - (W * Q)
         + x_G * (Q2 + R2) - y_G * P*Q - z_G * P*R)<< endl;

cout << "term2=" <<    + rho2 * L4 * (  X_pp * P2   + X_qq * Q2
                       + X_rr * R2   + X_pr * P*R)
     << endl;

cout << "term3=" <<    + rho2 * L3 * (  X_wq * W*Q + X_vp * V*P + X_vr * V*R

                       + U*Q * (  X_uq_delta_bow      * delta_planes_bow
                                + X_uq_delta_stern    * delta_planes_stern)

                       + U*R * (  X_ur_delta_rudder   * delta_rudder_stern
                                + X_ur_delta_rudder   * delta_rudder_bow)
                      )
     << endl;

cout << "term4=" <<    + rho2 * L2 * (  X_vv * V2   + X_ww * W2

                       + U*V * (  X_uv_delta_rudder   * delta_rudder_stern)

                       + U*W * (  X_uw_delta_bow      * delta_planes_bow
                                + X_uw_delta_stern    * delta_planes_stern)

                       + U * fabs (U)    * (   X_uu_delta_b_delta_b
```

```
                                                    * delta_planes_bow
                                                    * delta_planes_bow

                                                  + X_uu_delta_s_delta_s
                                                    * delta_planes_stern
                                                    * delta_planes_stern

                                                  + X_uu_delta_r_delta_r
                                                    * delta_rudder_bow
                                                    * delta_rudder_bow

                                                  + X_uu_delta_r_delta_r
                                                    * delta_rudder_stern
                                                    * delta_rudder_stern)
                         )
     << endl;

cout << "term5=" << - (Weight - Buoyancy) * sinTHETA
     << endl;

cout << "term6,term7=" <<   "EPSILON terms, no longer used"
     << endl;

// cout << "term6=" <<  rho2 * L3 *   X_qdsn  * U*Q * delta_planes_stern
//      * EPSILON << endl;
//
// cout << "term7=" <<  rho2 * L2 * EPSILON * ( X_wdsn  * U*W
//                                                       * delta_planes_stern
//                                  + X_dsdsn * U2       * delta_planes_stern
//                                                       * delta_planes_stern)
//      << endl;

cout << "term8=" <<   + rho2 * L2 * C_d0 *  square (speed_per_rpm)

                   *  0.5 * (  AUV_port_rpm * fabs (AUV_port_rpm)

                               + AUV_stbd_rpm * fabs (AUV_stbd_rpm))
     << endl;

cout << "term9=" << - rho2 * L2 * C_d0 * U * fabs (U)
     << endl;
}

////////////////////////////////////////////////////////////////////////////////

    rhs [SWAY ] = // Sway  Motion Equation right hand side -------------------//

         m * (- (U * R) + (W * P) - x_G * (P * Q)

                                  + y_G * (P2 + R2)

                                  - z_G * (Q  * R))

       + rho2 * L4 * (  Y_pq     * P*Q    + Y_qr    * Q*R)

       + rho2 * L3 * (  Y_up     * U*P    + Y_ur    * U*R

                      + Y_vq     * V*Q    + Y_wp    * W*P    +  Y_wr * W*R)

       + rho2 * L2 * (  Y_uv     * U*V    + Y_vw    * V*W

                      + U*fabs(U) * Y_uu_delta_rb * delta_rudder_bow

                      + U*fabs(U) * Y_uu_delta_rs * delta_rudder_stern)

       - sway_integral
```

```
      + (Weight - Buoyancy) * cosTHETA * sinPHI

      -  (2.0 / (24.0 * 24.0))  // each thruster 2.0 lb per 24V signal squared

          * (  AUV_bow_lateral    * fabs (AUV_bow_lateral)
             + AUV_stern_lateral * fabs (AUV_stern_lateral));

////////////////////////////////////////////////////////////////////////////////

if (TRACE)    // Sway TRACE
{
cout << "* sway term1=" <<  m * (- (U * R) + (W * P)
                                 - x_G * (P * Q)
                                 + y_G * (P2 + R2)
                                 - z_G * (Q  * R))
     << endl;

cout << "term2=" <<    + rho2 * L4 * (  Y_pq    * P*Q    + Y_qr   * Q*R)
     << endl;

cout << "term3=" <<    + rho2 * L3 * (  Y_up    * U*P    +  Y_ur  * U*R

                       + Y_vq    * V*Q    + Y_wp    * W*P    +  Y_wr * W*R)
     << endl;

cout << "term4=" <<    + rho2 * L2 * (  Y_uv     * U*V    + Y_vw   * V*W

                       + U*fabs(U) * Y_uu_delta_rb * delta_rudder_bow

                       + U*fabs(U) * Y_uu_delta_rs * delta_rudder_stern)
     << endl;

cout << "term5=" <<    - sway_integral
     << endl;

cout << "term6=" <<    + (Weight - Buoyancy) * cosTHETA * sinPHI
     << endl;

cout << "term7=" <<    -  (2.0 / (24.0 * 24.0))
                      // each thruster 2.0 lb per 24V signal squared

                       * (  AUV_bow_lateral    * fabs (AUV_bow_lateral)
                          + AUV_stern_lateral * fabs (AUV_stern_lateral))
     << endl;
}

////////////////////////////////////////////////////////////////////////////////

   rhs [HEAVE] = // Heave Motion Equation right hand side --------------------//

         m * (  (U * Q) - (V * P) - x_G * (P * R) - y_G * (Q * R)

                                                + z_G * (P2 + Q2))

      + rho2 * L4 * (  Z_pp    * P2     + Z_pr * P*R     + Z_rr * R2)

      + rho2 * L3 * (  Z_uq    * U*Q    + Z_vp * V*P     + Z_vr * V*R)

      + rho2 * L2 * (  Z_uw    * U*W    + Z_vv * V2

                      + ( U*fabs(U) * Z_uu_delta_b * delta_planes_bow  )

                      + ( U*fabs(U) * Z_uu_delta_s * delta_planes_stern))
```

```
        - heave_integral

        + (Weight - Buoyancy) * cosTHETA * cosPHI

        // EPSILON terms have been removed due to revised equations of motion

        // + rho2 * L3 *     Z_qn   * U*Q * EPSILON
        // + rho2 * L2 * (   Z_wn   * U*W
        //                 + Z_dsn * U*fabs(U) * delta_planes_stern) * EPSILON

        +  (2.0 / (24.0 * 24.0))  // each thruster 2.0 lb per 24V signal squared

           * ( AUV_bow_vertical   * fabs (AUV_bow_vertical)  +
               AUV_stern_vertical * fabs (AUV_stern_vertical));

////////////////////////////////////////////////////////////////////////////////

if (TRACE)    // Heave TRACE
{
cout <<  "* heave term1=" <<  m * (  (U * Q) - (V * P) - x_G * (P * R)
                                                         - y_G * (Q * R)
                                                         + z_G * (P2 + Q2))
     << endl;

cout << "term2=" <<    + rho2 * L4 * (  Z_pp    * P2     + Z_pr    * P*R
        + Z_rr * R2) << endl;

cout << "term3=" <<    + rho2 * L3 * (  Z_uq    * U*Q    + Z_vp * V*P
        + Z_vr * V*R) << endl;

cout << "term4=" <<    + rho2 * L2 * (  Z_uw    * U*W    + Z_vv  * V2

                         + ( U*fabs(U) * Z_uu_delta_b * delta_planes_bow  )

                         + ( U*fabs(U) * Z_uu_delta_s * delta_planes_stern))
     << endl;

cout << "term5=" <<    - heave_integral
     << endl;

cout << "term6=" <<    + (Weight - Buoyancy) * cosTHETA * cosPHI
     << endl;

cout << "term7=" <<    "EPSILON terms, no longer used"
     << endl;

cout << "term8=" <<    + rho2 * L2 * ( Z_wn   * U*W )
     << endl;

cout << "term9=" <<    +   (2.0 / (24.0 * 24.0))
         // each thruster 2.0 lb per 24V signal squared

                          * ( AUV_bow_vertical   * fabs (AUV_bow_vertical)  +
                              AUV_stern_vertical * fabs (AUV_stern_vertical) )
     << endl;
}
////////////////////////////////////////////////////////////////////////////////

   rhs [ROLL ] = // Roll  Motion Equation right hand side --------------------//

        - (I_z - I_y) * Q*R - I_xy * P*R + I_yz * (Q2 - R2) + I_xz * P*Q

        - m * ( y_G * ( -U*Q + V*P) - z_G  * ( U*R - W*P))

        + rho2 * L5  * (  K_pq * P*Q  + K_qr * Q*R
```

```
                    + K_pp * P * fabs(P)
                    + K_p  * P              )  // hovering roll drag

    + rho2 * L4   * ( K_up * fabs(U)*P   + K_ur * U*R   + K_vq * V*Q

                    + K_wp * W*P   + K_wr * W*R)

    + rho2 * L3   * (  K_uv * U*V   + K_vw * V*W

                    - U*fabs(U) * 0.5 * (  K_uu_planes * delta_planes_bow
                                         + K_uu_planes * delta_planes_stern))
    // expected: opposed plane directions ^ cause negation & cancellation

    + (y_G * Weight - y_B * Buoyancy) * cosTHETA * cosPHI

    - (z_G * Weight - z_B * Buoyancy) * cosTHETA * sinPHI;

    // EPSILON terms have been removed due to revised equations of motion
    // + rho2 * L4 * K_pn * U*P * EPSILON

    // + rho2 * L3 * U*fabs(U)   * K_prop; // oversimplified, in error

/////////////////////////////////////////////////////////////////////////////////

if (TRACE)    // Roll TRACE
{
cout << "* roll term1=" << - (I_z - I_y) * Q*R - I_xy * P*R + I_yz * (Q2 - R2)
                          + I_xz * P*Q
     << endl;

cout << "term2=" << - m * ( y_G * ( -U*Q + V*P) - z_G  * ( U*R - W*P))
     << endl;

cout << "term3=" << + rho2 * L5   * (  K_pq * P*Q   + K_qr * Q*R

                    + K_pp * P * fabs(P)
                    + K_p  * P              )  // hovering roll drag
     << endl;

cout << "term4=" << + rho2 * L4   * ( K_up   * fabs(U)*P   + K_ur * U*R

                    + K_vq * V*Q + K_wp * W*P   + K_wr * W*R)
     << endl;

cout << "term5=" << + rho2 * L3   * (  K_uv * U*V   + K_vw * V*W

                    - U*fabs(U) * 0.5 * (  K_uu_planes * delta_planes_bow
                                         + K_uu_planes * delta_planes_stern))
      // expected: opposed plane directions ^ cause negation & cancellation
     << endl;

cout << "term6=" << + (y_G * Weight - y_B * Buoyancy) * cosTHETA * cosPHI
     << endl;

cout << "term7=" << - (z_G * Weight - z_B * Buoyancy) * cosTHETA * sinPHI
     << endl;

cout << "term8,term9=" <<   "EPSILON terms, no longer used"
     << endl;

// cout << "term8=" << + rho2 * L4 * K_pn * U*P * EPSILON
//      << endl;

// cout << "term9=" << + rho2 * L3 * U*fabs(U)   * K_prop
//      << endl;
}
```

```
/////////////////////////////////////////////////////////////////////////////////

   rhs [PITCH] = // Pitch Motion Equation right hand side --------------------//

        - (I_x - I_z) * P*R + I_xy * Q*R - I_yz * P*Q - I_xz * (P2 - R2)

        +  m * ( x_G * ( -U*Q + V*P) - z_G  * ( - V*R + W*Q))

        + rho2 * L5  * (  M_pp * P2   + M_pr * P*R  + M_rr * R*fabs (R)

                          + M_q  * Q
                          + M_qq * Q * fabs(Q))  // hovering pitch drag

        + rho2 * L4  * ( M_uq * U*Q  + M_vp * V*P  + M_vr * V*R)

        + rho2 * L3  * ( M_uw * U*W  + M_vv * V2

                        + U*fabs(U) * (  M_uu_delta_bow   * delta_planes_bow
                                       + M_uu_delta_stern * delta_planes_stern))

        + pitch_integral  // note sign corrections to Healey pitch_integral

        - (x_G * Weight   -  x_B * Buoyancy) * cosTHETA * cosPHI

        - (z_G * Weight   -  z_B * Buoyancy) * sinTHETA

        +  (2.0 / (24.0 * 24.0))  // each thruster 2.0 lb per 24V signal squared
                                  // multiplied by respective moment arms
                                  // x_bow_vertical (+), x_stern_vert (-)

        * ( (AUV_bow_vertical   * fabs (AUV_bow_vertical)   * x_bow_vertical)
           +(AUV_stern_vertical * fabs (AUV_stern_vertical) * x_stern_vertical));

        // EPSILON terms have been removed due to revised equations of motion
        // + rho2 * L4  *  M_qn * U*Q * EPSILON
        // + rho2 * L3  * (M_wn * U*W + M_dsn * U*fabs(U) * delta_planes_stern)
        //                * EPSILON;

/////////////////////////////////////////////////////////////////////////////////

if (TRACE)    // Pitch TRACE
{
cout << "* pitch term1=" << - (I_x - I_z) * P*R + I_xy * Q*R - I_yz * P*Q
        - I_xz * (P2 - R2) << endl;

cout << "term2=" << +  m * ( x_G * ( -U*Q + V*P) - z_G  * ( - V*R + W*Q))
     << endl;

cout << "term3=" << + rho2 * L5  * (  M_pp * P2   + M_pr * P*R  + M_rr
                                                                 * R*fabs (R)
                        + M_q  * Q
                        + M_qq * Q * fabs(Q))  // hovering pitch drag
     << endl;

cout << "term4=" << + rho2 * L4  * ( M_uq * U*Q  + M_vp * V*P  + M_vr * V*R)
     << endl;

cout << "term5=" << + rho2 * L3  * ( M_uw * U*W  + M_vv * V2

                      + U*fabs(U) * (  M_uu_delta_bow   * delta_planes_bow
                                     + M_uu_delta_stern * delta_planes_stern))
     << endl;

cout << "term6=" << + pitch_integral
     << endl;

cout << "term7=" << - (x_G * Weight   -  x_B * Buoyancy) * cosTHETA * cosPHI
```

```
     << endl;

cout << "term8=" << - (z_G * Weight   -  z_B * Buoyancy) * sinTHETA
     << endl;

cout << "term9=" << +  (2.0 / (24.0 * 24.0))
                                  // each thruster 2.0 lb per 24V signal squared
                                  // multiplied by respective moment arms
                                  // x_bow_vertical (+). x_stern_vert (-)

        * ( (AUV_bow_vertical   * fabs (AUV_bow_vertical)   * x_bow_vertical)
          +(AUV_stern_vertical * fabs (AUV_stern_vertical) * x_stern_vertical))
     << endl;

cout << "term10,term11=" <<   "EPSILON terms, no longer used"
     << endl;

// cout << "term10=" << + rho2 * L4  *  M_qn * U*Q * EPSILON
//        << endl;

// cout << "term11=" << + rho2 * L3  * (M_wn * U*W + M_dsn * U*fabs(U)
//                                                       * delta_planes_stern)
//                        * EPSILON
//      << endl;
}

////////////////////////////////////////////////////////////////////////////////

  rhs [YAW  ] = // Yaw  Motion Equation right hand side --------------------//

      - (I_y - I_x) * P*Q + I_xy * (P2 - Q2) + I_yz * P*R - I_xz * Q*R

      -  m * ( x_G * ( U*R - W*P) - y_G   * ( - V*R + W*Q))

      + rho2 * L5  * (  N_pq * P*Q  + N_qr * Q*R

                        + N_r  * R
                        + N_rr * R * fabs (R))  // hovering yaw drag

      + rho2 * L4  * ( N_up * U*P  + N_ur * U*R  + N_vq * V*Q

                       + N_wp * W*P  + N_wr * W*R)

      + rho2 * L3  * ( N_uv * U*V  + N_vw * V*W

                       + U*fabs(U) * N_uu_delta_rb * delta_rudder_bow
                       - U*fabs(U) * N_uu_delta_rs * delta_rudder_stern)

      - yaw_integral

      + (x_G * Weight  -  x_B * Buoyancy) * cosTHETA * sinPHI

      + (y_G * Weight  -  y_B * Buoyancy) * sinTHETA

      -  (2.0 / (24.0 * 24.0)) // each thruster 2.0 lb per 24V signal squared
                               // multiplied by respective moment arms
        * ( (AUV_bow_lateral   * fabs (AUV_bow_lateral)   * x_bow_lateral    )

          +(AUV_stern_lateral * fabs (AUV_stern_lateral) * x_stern_lateral ))

      - rho2 * L2 * C_d0

                * ( square (speed_per_rpm) *  0.5 // propeller yaw

                    * (  AUV_port_rpm * fabs(AUV_port_rpm) * y_port_propeller

                       + AUV_stbd_rpm * fabs(AUV_stbd_rpm) * y_stbd_propeller)
```

```
                    - U * fabs(U));

////////////////////////////////////////////////////////////////////////////////

if (TRACE)    // Yaw TRACE
{
cout << "* yaw term1=" <<  - (I_y - I_x) * P*Q + I_xy * (P2 - Q2)
                              + I_yz * P*R - I_xz * Q*R
     << endl;

cout << "term2=" <<     -  m * ( x_G * ( U*R - W*P) - y_G    * ( - V*R + W*Q))
     << endl;

cout << "term3=" <<    + rho2 * L5   * (  N_pq * P*Q  + N_qr * Q*R

                        + N_r   * R
                        + N_rr * R * fabs (R))  // hovering yaw drag
     << endl;

cout << "term4=" <<    + rho2 * L4   * ( N_up * U*P  + N_ur * U*R  + N_vq * V*Q

                       + N_wp * W*P  + N_wr * W*R)
     << endl;

cout << "term5=" << + rho2 * L3   * ( N_uv * U*V  + N_vw * V*W

                       + U*fabs(U) * N_uu_delta_rb * delta_rudder_bow
                       - U*fabs(U) * N_uu_delta_rs * delta_rudder_stern)
     << endl;

cout << "term6=" <<  - yaw_integral
     << endl;

cout << "term7=" <<  + (x_G * Weight   -  x_B * Buoyancy) * cosTHETA * sinPHI
     << endl;

cout << "term8=" <<  + (y_G * Weight   -  y_B * Buoyancy) * sinTHETA
     << endl;

cout << "term9=" << -  (2.0 / (24.0 * 24.0))
                        // each thruster 2.0 lb per 24V signal squared
                        // multiplied by respective moment arms
          * ( (AUV_bow_lateral   * fabs (AUV_bow_lateral)   * x_bow_lateral    )

            +(AUV_stern_lateral * fabs (AUV_stern_lateral) * x_stern_lateral ))
     << endl;

cout << "term10=" << - rho2 * L2 * C_d0

                 * ( square (speed_per_rpm) *  0.5 // propeller yaw

                    * (  AUV_port_rpm * fabs(AUV_port_rpm) * y_port_propeller

                       + AUV_stbd_rpm * fabs(AUV_stbd_rpm) * y_stbd_propeller)

                    - U * fabs(U))
     << endl;
}

////////////////////////////////////////////////////////////////////////////////

// debug:

if (TRACE)
{
   cout << "     SURGE  = " <<     SURGE  << endl;
   cout << "     SWAY   = " <<     SWAY   << endl;
```

```
   cout << "      HEAVE   = " <<       HEAVE   << endl;
   cout << "      ROLL    = " <<       ROLL    << endl;
   cout << "      PITCH   = " <<       PITCH   << endl;
   cout << "      YAW     = " <<       YAW     << endl;

   cout << "rhs [SURGE] = " << rhs [SURGE] << endl;
   cout << "rhs [SWAY ] = " << rhs [SWAY ] << endl;
   cout << "rhs [HEAVE] = " << rhs [HEAVE] << endl;
   cout << "rhs [ROLL ] = " << rhs [ROLL ] << endl;
   cout << "rhs [PITCH] = " << rhs [PITCH] << endl;
   cout << "rhs [YAW  ] = " << rhs [YAW  ] << endl;

// cout << "mass_inverse:" << endl;   print_matrix6x6 (mass_inverse);

   cout << "velocities:      <" << U << ", " << V << ", " << W << ", "
                                << P << ", " << Q << ", " << R << ">" << endl;

   cout << "RHS:              ";         print_matrix6 (rhs);

// rhs [SURGE] = 0.0;
// rhs [SWAY ] = 0.0;
// rhs [HEAVE] = 0.0;
// rhs [ROLL ] = 0.0;
// rhs [PITCH] = 0.0;
// rhs [YAW  ] = 0.0;

//////////////////////////////////////////////////////////////////////////////

   // calculate new accelerations matrix using mass_inverse & rhs, print -----//

   multiply6x6_6 (mass_inverse, rhs, new_acceleration);

   if (TRACE)
   {
      cout << "Accelerations:  ";   print_matrix6 (new_acceleration);
   }

   // limit accelerations ---------------------------------------------------//

   clamp(& new_acceleration [SURGE], -5.0, 5.0, "new_acceleration [SURGE]");
   clamp(& new_acceleration [SWAY ], -5.0, 5.0, "new_acceleration [SWAY ]");
   clamp(& new_acceleration [HEAVE], -5.0, 5.0, "new_acceleration [HEAVE]");

   clamp(& new_acceleration [ROLL ], -5.0, 5.0, "new_acceleration [ROLL ]");
   clamp(& new_acceleration [PITCH], -5.0, 5.0, "new_acceleration [PITCH]");
   clamp(& new_acceleration [YAW  ], -5.0, 5.0, "new_acceleration [YAW  ]");


   // find velocities by integrating averaged accelerations -----------------//
   //        (Heun integration)

   new_velocity [SURGE] = 0.5 * (u_dot + new_acceleration [SURGE]) * dt + U;
   new_velocity [SWAY ] = 0.5 * (v_dot + new_acceleration [SWAY ]) * dt + V;
   new_velocity [HEAVE] = 0.5 * (w_dot + new_acceleration [HEAVE]) * dt + W;
   new_velocity [ROLL ] = 0.5 * (p_dot + new_acceleration [ROLL ]) * dt + P;
   new_velocity [PITCH] = 0.5 * (q_dot + new_acceleration [PITCH]) * dt + Q;
   new_velocity [YAW  ] = 0.5 * (r_dot + new_acceleration [YAW  ]) * dt + R;

   // find velocities by integrating instantaneous accelerations
   //        (this method is less accurate and is not used, although at small
   //         timesteps the difference is negligible)
   //        (Euler integration)

   // new_velocity [SURGE] = (new_acceleration [SURGE]) * dt + U;
   // new_velocity [SWAY ] = (new_acceleration [SWAY ]) * dt + V;
```

```
// new_velocity [HEAVE] = (new_acceleration [HEAVE]) * dt + W;
// new_velocity [ROLL ] = (new_acceleration [ROLL ]) * dt + P;
// new_velocity [PITCH] = (new_acceleration [PITCH]) * dt + Q;
// new_velocity [YAW  ] = (new_acceleration [YAW  ]) * dt + R;

// note that surge velocity may be negative under to model constraints
// but this is a problem so it is clamped to be non-negative

clamp (& new_velocity [SURGE], -3.0, 3.0, "new_velocity [SURGE] velocity");
                                // 3.0 is set arbitrarily high

// update UUVBody state accelerations to newly-calculated values   --------//

u_dot = new_acceleration [SURGE];
v_dot = new_acceleration [SWAY ];
w_dot = new_acceleration [HEAVE];
p_dot = new_acceleration [ROLL ];
q_dot = new_acceleration [PITCH];
r_dot = new_acceleration [YAW  ];

// calculate world coordinate system linear & angular velocities   --------//

//  see Cooke Figure 10 for corrections to Healey equations for x/y/z_dot:
//  also Healey course notes eqn (26) and Frank-McGhee corrected paper (A.8)

x_dot = AUV_oceancurrent_u

      + U *  cos (PSI) * cos (THETA)

      + V * (cos (PSI) * sin (THETA) * sin (PHI) - sin (PSI) * cos(PHI))

      + W * (cos (PSI) * sin (THETA) * cos (PHI) + sin (PSI) * sin(PHI));

y_dot = AUV_oceancurrent_v

      + U *  sin (PSI) * cos (THETA)

      + V * (sin (PSI) * sin (THETA) * sin (PHI) + cos (PSI) * cos(PHI))

      + W * (sin (PSI) * sin (THETA) * cos (PHI) - cos (PSI) * sin(PHI));

z_dot = AUV_oceancurrent_w

      - U * sin (THETA)

      + V * cos (THETA) * sin (PHI)

      + W * cos (THETA) * cos (PHI);


phi_dot   = P + Q * sin (PHI) * tan (THETA)

              + R * cos (PHI) * tan (THETA);

theta_dot =     Q * cos (PHI)

              - R * sin (PHI);


if  (cos (THETA) == 0.0)
{
     cout << "UUVBody::integrate_equations_of_motion ():  "        << endl;
     cout << "   cos (THETA) == 0.0 so psi_dot set equal to zero." << endl;
     psi_dot = 0.0;
}
else psi_dot = (Q * sin (PHI) + R * cos (PHI)) / cos (THETA);
```

```
Vector3D linear_rates =  Vector3D (x_dot, y_dot, z_dot);
if (TRACE)
{
     cout << endl;
     cout << "<x_dot, y_dot, z_dot>          = " << linear_rates << endl;
     cout << "                      magnitude = " << linear_rates.magnitude ()
          << endl;
}

Vector3D euler_rates =  Vector3D (phi_dot, theta_dot, psi_dot);
if (TRACE)
{
     cout << "<phi_dot, theta_dot, psi_dot> = " << euler_rates << endl;
     cout << "                      magnitude = " << euler_rates.magnitude ()
          << endl;
}

// calculate world coordinate system homogenous transform matrix   --------//

Hmatrix Hincremental = Hmatrix (); // default initialization
Hincremental.set_posture ( P * dt, Q * dt, R * dt );
Hincremental.rotate      ( PHI,    THETA,  PSI    );
double w_x = Hincremental.phi_value   ();
double w_y = Hincremental.theta_value ();
double w_z = Hincremental.psi_value   ();


Vector3D world_rates =  Vector3D (w_x, w_y, w_z);
if (TRACE)
{
     cout << "<w_x, w_y, w_z>                = " << world_rates << endl;
     cout << "                      magnitude = " << world_rates.magnitude ()
          << endl;
}

Hmatrix Hrevised1 = Hmatrix ();   // default initialization
Hrevised1.incremental_rotation    ( phi_dot, theta_dot, psi_dot, dt );
Hrevised1.incremental_translation ( U, V, W, dt );

Hmatrix Hproduct1 = Hprevious * Hrevised1;

Hmatrix Hrevised2 = Hmatrix (   x_dot * dt,     y_dot * dt,   z_dot * dt,
                              phi_dot * dt, theta_dot * dt, psi_dot * dt);

Hprevious = Hproduct1;

// translate and rotate and update time in RigidBody state ----------------//
// note world coordinate system is used by RigidBody:

set_angular_velocities (phi_dot, theta_dot, psi_dot);

set_linear_velocities  (  x_dot,     y_dot,    z_dot);

set_time_of_posture    (current_uuv_time);

update_Hmatrix         (dt);

if (TRACE)
{
    cout << "incremental hmatrix = ";
    Hincremental.print_hmatrix ();
    cout << "revised1 hmatrix = ";
    Hrevised1.print_hmatrix ();
    cout << "revised2 hmatrix = ";
    Hrevised2.print_hmatrix ();
    cout << "product1 hmatrix = ";
```

```
        Hproduct1.print_hmatrix ();

        cout << "original hmatrix = ";
        hmatrix.print_hmatrix ();
    }

    cout << "substituting product1 hmatrix" << endl;
    hmatrix = Hproduct1;

    // --------------------------------------------------------------------

    // Save body-coordinate-system velocities for the next loop:

    U = new_velocity [SURGE];
    V = new_velocity [SWAY ];
    W = new_velocity [HEAVE];
    P = new_velocity [ROLL ];
    Q = new_velocity [PITCH];
    R = new_velocity [YAW  ];

//  cout << "world U =" << U << ", x_dot     = " << x_dot     << endl;
//  cout << "world V =" << V << ", y_dot     = " << y_dot     << endl;
//  cout << "world W =" << W << ", z_dot     = " << z_dot     << endl;
//  cout << "world P =" << P << ", phi_dot   = " << phi_dot   << endl;
//  cout << "world Q =" << Q << ", theta_dot = " << theta_dot << endl;
//  cout << "world R =" << R << ", psi_dot   = " << psi_dot   << endl;

    // --------------------------------------------------------------------
    // update all hydrodynamics-model-provided state variables in AUV_globals.h
    //         prior to retransmittal to AUV via AUVsocket

    AUV_time      = current_uuv_time; // mission time

    AUV_x         = x_value     (); //  x     position in world coordinates
    AUV_y         = y_value     (); //  y     position in world coordinates
    AUV_z         = z_value     (); //  z     position in world coordinates
    AUV_phi       = phi_value   (); //  roll   posture in world coordinates
    AUV_theta     = theta_value (); //  pitch  posture in world coordinates
    AUV_psi       = psi_value   (); //  yaw    posture in world coordinates

    AUV_u    = new_velocity [SURGE]; //  surge  linear velocity along x-axis
    AUV_v    = new_velocity [SWAY ]; //  sway   linear velocity along y-axis
    AUV_w    = new_velocity [HEAVE]; //  heave  linear velocity along x-axis
    AUV_p    = new_velocity [ROLL ]; //  roll  angular velocity about x-axis
    AUV_q    = new_velocity [PITCH]; //  pitch angular velocity about y-axis
    AUV_r    = new_velocity [YAW  ]; //  yaw   angular velocity about z-axis

    AUV_u_dot     = u_dot;           //      linear acceleration along x-axis
    AUV_v_dot     = v_dot;           //      linear acceleration along y-axis
    AUV_w_dot     = w_dot;           //      linear acceleration along x-axis
    AUV_p_dot     = p_dot;           //     angular acceleration about x-axis
    AUV_q_dot     = q_dot;           //     angular acceleration about y-axis
    AUV_r_dot     = r_dot;           //     angular acceleration about z-axis

    AUV_x_dot     = x_dot;           //       Euler velocity along North-axis
    AUV_y_dot     = y_dot;           //       Euler velocity along  East-axis
    AUV_z_dot     = z_dot;           //       Euler velocity along Depth-axis
    AUV_phi_dot   = phi_dot;         // Euler rotation rate about North-axis
    AUV_theta_dot = theta_dot;       // Euler rotation rate about  East-axis
    AUV_psi_dot   = psi_dot;         // Euler rotation rate about Depth-axis
}

#undef UDOT
#undef VDOT
#undef WDOT
#undef PDOT
#undef QDOT
```

```
#undef RDOT
#undef SURGE
#undef SWAY
#undef HEAVE
#undef ROLL
#undef PITCH
#undef YAW

//---------------------------------------------------------------------------//

#endif  // UUVBODY_C
```

F. *AUVsocket.C* Communications with a Networked AUV

```
/*******************************************************************************
 Program:            AUVsocket.C

 Description:        socket from auv execution level to Irix auv virtual world
                     receives partial telemetry, returns full telemetry with
                          dynamics parameters added

                     AUVsocket.C is a function library used by UUVBody

                     AUVsocket also provides voice server interface

 Revised:            28 October 94

 Compilation:        unix> make AUVsocket

 Original bases:     os9server.c, dynamics.c

 References:     (1) (Gespac-provided) EVIRA EVLAN-11 Ethernet Data Link
                     Controller for the G64/G96 Bus/EVTCP Internet Package
                     technical manuals
                 (2) Internetworking with TCP/IP Volume I:  Principles,
                     Protocols and Architectures, Douglas E. Comer,
                     Prentice Hall, Englewood Cliffs NJ, 1991
                 (3) Internetworking with TCP/IP Volume II:  Design,
                     Implementation and Internals, Douglas E. Comer and
                     David L. Stevens, Prentice Hall, Englewood Cliffs NJ, 1991
                 (4) IRIX Network Programming Guide, Silicon Graphics Inc.
                 (5) An Advanced 4.3BSD Interprocess Communication Tutorial,
                     Samuel J. Leffler, Robert S. Fabry, William N. Joy,
                     Phil Lapsley, Steve Miller and Chris Torek, undated
                 (6) Real-Time Programming Tutorial, Bill Mannel, SGI Expo,
                     Silicon Graphics Inc., 23 May 93
                 (7) "Say..."  Axel Belinfante's speech server at University
                     of Twente, Netherlands, which also uses Nick Ing-Simmons'
                     phoneme synthesizer 'rsynth':
                     http://utis179.cs.utwente.nl:8001/say/?

 Status:             Tested satisfactorily


 Future work:        Consider implementation as an inetd 'superserver' daemon or
                         just closing socket when done, reopening & waiting

                     Consider bounds diagnostic checking on values transferred
                         over socket

                     Fix type mismatch on signal () calls - very gnarly problem!

                     Comments and suggestions are welcome!
*/

/*******************************************************************************/

#ifndef AUVSOCKET_C
#define AUVSOCKET_C     /* prevent errors if multiple #includes present */

/*******************************************************************************/

#define _BSD_SIGNALS
#include <ctype.h>
#include <signal.h>
#include <stdio.h>
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>

#include <time.h>
#include <stdlib.h>
#include <unistd.h>

#include "AUVglobals.H"   // global state vector
#include "Quaternion.C"   // degrees/radians conversions

// extern "C" entries provide function compatibility with C++ compiler - - - - -

extern "C" {
                void  bzero   (void *b, int length);
                void  bcopy   (const void *src, void *dst, int length);
                int   strncmp (const char *s1, const char *s2, size_t n);
                char *strcpy  (char *s1, const char *s2);
                char *strncat (char *s1, const char *s2, size_t n);

                //  reference p. 434 Kelley & Pohl "A Book on C",
                //     /usr/include/sys/signal.h and /usr/include/signal.h

                // int (*signal(int,int (*)(int, ...)))(int, ...);
           };

/****************************************************************************/

/* One stream socket is used with adequate throughput                        */
/*   (although two could work, no performance improvement is expected)       */

/* Be careful that you reserve these port numbers to prevent collisions     */
/*    from other processes requesting ports on your system:                  */

#define DISBRIDGE_TCP_PORT     2056 /* disbridge 1.3 program, server & client */

#define NPSNET_MCAST_PORT      3111 /* Mike Macedonia's multicast DIS 2.0.3   */

                                    /* NPS Autonomous Underwater Vehicle (AUV)*/
                                    /*     Underwater Virtual World (UVW)     */

#define AUVSIM1_TCP_PORT_0     3210 /* os9sender <==> os9server test programs */
#define AUVSIM1_TCP_PORT_1     3211 /* auv execution level <==> virtual world */
#define AUVSIM1_TCP_PORT_2     3212 /* auv execution <==> tactical (networked)*/
#define AUVSIM1_TCP_PORT_3     3213 /*        port for future use             */
#define AUVSIM1_TCP_PORT_4     3214 /*        port for future use             */
#define AUVSIM1_TCP_PORT_5     3215 /*        port for future use             */
#define AUVSIM1_TCP_PORT_6     3216 /*        port for future use             */
#define AUVSIM1_TCP_PORT_7     3217 /*        port for future use             */
#define AUVSIM1_TCP_PORT_8     3218 /*        port for future use             */
#define AUVSIM1_TCP_PORT_9     3219 /*        port for future use             */

#define SOCKET_QUEUE_SIZE   5       /* max allowed by TCP/IP                  */

#define AUVSIM_EXECUTION_LEVEL_HOST_NAME "auvsim1.or.nps.navy.mil"

/****************************************************************************/

/* function prototypes *******************************************************/

   int     open_execution_level_socket      ();

   int     shutdown_socket                  (int signal_thrown);

   int     read_from_execution_level_socket ();  // returns -1 on failure
```

```
void    write_to_execution_level_socket   ();

double  normalize                         (double degs);

double  normalize2                        (double degs);

void    AUV_global_initialization         ();

/*****************************************************************************/

/* global variable definitions ***********************************************/

static int                   socket_descriptor,
                             socket_accepted,
                             socket_stream;

static int                   socket_length = 255;  /* max allowed packet size */

static int                   bytes_received, bytes_read, bytes_written,
                                             bytes_left, bytes_sent;

static char                  command_sent     [300],
                             command_received [300],
                             command_copy     [300],
                             remote_host_name [60];

static int                   shutdown_signal_received = FALSE;

static FILE                * checksoundfile;

static struct sockaddr_in server_address;

static struct hostent      * server_entity;

static char                  keyword [81];

static char                * ptr_index;
static char                * remote_buffer;

static int                   state_variables_received;
static int                   return_state_vector;
static char                  www_execution_message_string [300];
static char                  answer;

static int                   socket_already_opened = FALSE;
static int                   audio_enabled         = TRUE;
/*****************************************************************************/

int open_execution_level_socket ()        /* Initialize communications blocks */
{
      if (socket_already_opened == TRUE)
      {
            if (TRACE)
            {
              cout << "open_execution_level_socket ():  socket_already_opened,"
                   << " returning" << endl;
            }
            return (-1);
      }
      else if (TRACE) cout << "open_execution_level_socket () starting" << endl;

/*****************************************************************************/
/* Initialize both client & server *******************************************/

   /* Signal handlers for termination to override net_open () and net_close ()*/
```

```
   /*    signal handlers.  Otherwise you are unable to ^C kill this program.  */

#if defined(sgi)
// signal (SIGHUP,  shutdown_socket);             /* hangup                      */
// signal (SIGINT,  shutdown_socket);             /* interrupt character         */
// signal (SIGKILL, shutdown_socket);             /* kill signal from Unix       */
// signal (SIGPIPE, shutdown_socket);             /* broken pipe from other host */
// signal (SIGTERM, shutdown_socket);             /* software termination        */
#endif

/*************************************************************************/
/* Initialize server *****************************************************/

   /* setup to listen for client to attempt connection                    */
   {
      /* Server opens server port ********************************************/

      /* Open TCP (Internet stream) in socket                             */
      if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
      {
          cout << "open_execution_level_socket () can't 'open' stream socket"
               << endl;
          return (-1);
      }
      else if (TRACE)
           {
             cout << "open_execution_level_socket () socket 'open' successful"
                  << endl;
           }

      /* Bind local address so client can talk to server                 */
#if defined(sgi)
      bzero ((void *) &server_address, sizeof (server_address));
#endif
      server_address.sin_family      = AF_INET;   /* Internet protocol family */

      /* make sure port is in network byte order                          */
      server_address.sin_addr.s_addr = htonl (INADDR_ANY);
      server_address.sin_port        = htons (AUVSIM1_TCP_PORT_1);

      if (bind (                    socket_descriptor,
                (struct sockaddr *) &server_address,
                            sizeof (server_address)) < 0)
      {
          cout << "open_execution_level_socket () socket 'bind' unsuccessful"
               << endl;
          return (-1);
      }
      else if (TRACE)
           {
             cout << "open_execution_level_socket () socket 'bind' successful"
                  << endl;
           }

      /* prepare socket queue for connection requests using listen        */

      listen (socket_descriptor, SOCKET_QUEUE_SIZE);
      if (TRACE)
      {
        cout << "open_execution_level_socket () socket 'listen' complete ..."
             << endl;
      }

      /* Server 'accept' waits for client connections ************************/

      if (TRACE)
      {
```

```
      cout << "open_execution_level_socket () socket waiting to 'accept' ..."
           << endl;
    }
    bytes_received = sizeof (socket_descriptor);
    while ((socket_accepted  = accept ( socket_descriptor,
                                       &server_address,
                                       &bytes_received)) < 1)
       if (TRUE)  /* blocking code */
       {
         cout << "open_execution_level_socket () socket "
              << "'accept' unsuccessful, sleeping 1 second ..." << endl;
         sleep (1);
       }
    cout << "open_execution_level_socket () connection is open between "
         << "networks." << endl;

  } /* end initialization                                                */

  socket_stream = socket_accepted;    /* server */

  socket_already_opened = TRUE;

  if (TRACE)
  {
    cout << "AUVsocket SERVER: socket_descriptor = " << socket_descriptor
         << endl;
    cout << "                  socket_accepted   = " << socket_accepted
         << endl;
    cout << "                  socket_stream     = " << socket_stream
         << endl;
  }

  return (TRUE);
}

/***************************************************************************/
/***************************************************************************/

int read_from_execution_level_socket ()  // using global variables state vector
{
  /* temporary hold variables */

  double   AUV_time_temp,
           AUV_x_temp,               AUV_y_temp,              AUV_z_temp,
           AUV_phi_temp,             AUV_theta_temp,          AUV_psi_temp,
           AUV_u_temp,               AUV_v_temp,              AUV_w_temp,
           AUV_p_temp,               AUV_q_temp,              AUV_r_temp,
           AUV_x_dot_temp,           AUV_y_dot_temp,          AUV_z_dot_temp,
           AUV_phi_dot_temp,         AUV_theta_dot_temp,      AUV_psi_dot_temp,
           AUV_delta_rudder_temp,    AUV_delta_planes_temp,
           AUV_port_rpm_temp,        AUV_stbd_rpm_temp,
           AUV_bow_vertical_temp,    AUV_stern_vertical_temp,
           AUV_bow_lateral_temp,     AUV_stern_lateral_temp,
           AUV_ST1000_range_temp,    AUV_ST725_range_temp,
           AUV_ST1000_bearing_temp,  AUV_ST725_bearing_temp,
           AUV_ST1000_strength_temp,AUV_ST725_strength_temp;
  int      start_index, offset, index;

  /*************************************************************************/
  /* Receiver block                                                        */
  /*************************************************************************/

  for (index = 0; index < 300; index++)      /* uppercase  */
       command_received [index] = (char) 0;


  /* listen to remote host, relay to local network/program                 */
```

```
bytes_left          = socket_length;
bytes_received      = 0;
ptr_index           = command_received;

while ((bytes_left > 0) && (bytes_received >= 0))  /* read loop ************/
{
   bytes_read = read (socket_stream, ptr_index, bytes_left);

   if      (bytes_read <  0) bytes_received = bytes_read;
   else if (bytes_read >  0)
   {
        bytes_left     -= bytes_read;
        bytes_received += bytes_read;
        ptr_index      += bytes_read;
   }
   if (TRACE)
   {
        cout << "read_from_execution_level_socket () ";
        cout << "receiver block loop bytes_read = " << bytes_read<< endl;
   }

   /* if nothing is waiting to be read, break out of read loop               */
   if ((bytes_read == 0) && (bytes_received == 0)) break;
}

if  (bytes_received < 0)         /* failure     */
{
     if (TRACE)
     {
          cout << "read_from_execution_level_socket () ";
          cout << "receiver block read failed, bytes_received = "
               <<  bytes_received << endl;
     }
     return_state_vector = FALSE;
     return (return_state_vector);
}
else if (bytes_received == 0)  /* no transfer */
{
     if (TRACE)
     {
          cout << "read_from_execution_level_socket () ";
          cout << "receiver block bytes_received =  0 bytes" << endl;
     }
     return_state_vector = FALSE;
     return (return_state_vector);
}

/*  (bytes_received > 0) => socket read OK, print result if valid & traced */

if (TRUE)  // print telemetry record & key
{
   cout << command_received << endl;
   cout << "keyword,t,x,y,z,phi,theta,psi,u,v,w,p,q,r,"
        << "x_dot,y_dot,z_dot,phi_dot,theta_dot,psi_dot,"
        << "delta_rudder,delta_planes,l_rpm,r_rpm,"
        << "bow_vertical,bow_lateral,"
        << "stern_vertical,stern_lateral,"
        << "ST1000_bearing/range/strength,"
        << "ST725_bearing/range/strength" << endl;
}

state_variables_received = sscanf (command_received,
         "%s %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf
%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf \n",
          keyword,                  &AUV_time_temp,
         &AUV_x_temp,               &AUV_y_temp,                  &AUV_z_temp,
         &AUV_phi_temp,             &AUV_theta_temp,              &AUV_psi_temp,
```

```
            &AUV_u_temp,              &AUV_v_temp,              &AUV_w_temp,
            &AUV_p_temp,              &AUV_q_temp,              &AUV_r_temp,
            &AUV_x_dot_temp,          &AUV_y_dot_temp,          &AUV_z_dot_temp,
            &AUV_phi_dot_temp,        &AUV_theta_dot_temp,      &AUV_psi_dot_temp,
            &AUV_delta_rudder_temp,   &AUV_delta_planes_temp,
            &AUV_port_rpm_temp,       &AUV_stbd_rpm_temp,
            &AUV_bow_vertical_temp,   &AUV_stern_vertical_temp,
            &AUV_bow_lateral_temp,    &AUV_stern_lateral_temp,
            &AUV_ST1000_bearing_temp,&AUV_ST1000_range_temp,
                                      &AUV_ST1000_strength_temp,
            &AUV_ST725_bearing_temp,  &AUV_ST725_range_temp,
                                      &AUV_ST725_strength_temp);

for (index = 0; index < strlen (keyword); index++)      /* uppercase  */
      keyword [index] = toupper (keyword [index]);

if (TRACE) printf ("[AUVsocket keyword=%s]\n", keyword);

if (state_variables_received == 34)  // transfer was OK so keep new values
{
   if (TRACE) printf ("[AUVsocket state_variables_received == 34]\n");

        return_state_vector = TRUE;
        AUV_time              =              AUV_time_temp;
        AUV_x                 =              AUV_x_temp;
        AUV_y                 =              AUV_y_temp;
        AUV_z                 =              AUV_z_temp;
        AUV_phi               =    radians (AUV_phi_temp);
        AUV_theta             =    radians (AUV_theta_temp);
        AUV_psi               =    radians (AUV_psi_temp);
        AUV_u                 =              AUV_u_temp;
        AUV_v                 =              AUV_v_temp;
        AUV_w                 =              AUV_w_temp;
        AUV_p                 =    radians (AUV_p_temp);
        AUV_q                 =    radians (AUV_q_temp);
        AUV_r                 =    radians (AUV_r_temp);
        AUV_x_dot             =              AUV_x_dot_temp;
        AUV_y_dot             =              AUV_y_dot_temp;
        AUV_z_dot             =              AUV_z_dot_temp;
        AUV_phi_dot           =    radians (AUV_phi_dot_temp);
        AUV_theta_dot         =    radians (AUV_theta_dot_temp);
        AUV_psi_dot           =    radians (AUV_psi_dot_temp);
        AUV_delta_rudder      =    radians (AUV_delta_rudder_temp);
        AUV_delta_planes      =    radians (AUV_delta_planes_temp);
        AUV_port_rpm          =              AUV_port_rpm_temp;
        AUV_stbd_rpm          =              AUV_stbd_rpm_temp;
        AUV_bow_vertical      =              AUV_bow_vertical_temp;
        AUV_stern_vertical    =              AUV_stern_vertical_temp;
        AUV_bow_lateral       =              AUV_bow_lateral_temp;
        AUV_stern_lateral     =              AUV_stern_lateral_temp;
        AUV_ST1000_bearing    =              AUV_ST1000_bearing_temp;
        AUV_ST1000_range      =              AUV_ST1000_range_temp;
        AUV_ST1000_strength   =              AUV_ST1000_strength_temp;
        AUV_ST725_bearing     =              AUV_ST725_bearing_temp;
        AUV_ST725_range       =              AUV_ST725_range_temp;
        AUV_ST725_strength    =              AUV_ST725_strength_temp;
}
else  /* a message was received instead of telemetry                    */
{
   if (TRACE) printf ("[AUVsocket non-telemetry received]\n");

   return_state_vector = FALSE;

   if (strcmp (keyword, "AUV_STATE") == 0)
   {
        printf (
         "Garbled telemetry record received !!! variables_received=%d\n%s\n",
```

```
                    state_variables_received, command_received);
      fflush (stdout);
      return_state_vector      = FALSE;
      TRACE = TRUE;
      cin.get (answer); // pause
      cout << endl;
  }
  else if (strcmp (keyword, "MAIL") == 0)       /* Send e-mail request */
  {
     printf ("%s%n", command_received);
     system (         command_received);
  }
  else if (strncmp (command_received, "KILL", 4) == 0)
                                                /* KILL prior to reloop*/
  {
     if (TRACE) printf ("[AUVsocket KILL]\n");

     shutdown_socket (0);
     return_state_vector = -3;
  }
  else if (strcmp (keyword, "TIME") == 0)
  {
     if (TRACE) printf ("[AUVsocket TIME]\n");

     state_variables_received = sscanf (command_received, "%s%lf",
                                         keyword,
                                        &AUV_time_temp);

     if (state_variables_received == 2)  // transfer OK, keep new values
     {
       AUV_time             = AUV_time_temp;
       return_state_vector = FALSE;
       printf ("%s\n", command_received);
     }
     else
     {
       printf ("a bad time command was received & ignored:  %s",
                                                          command_received);
       return_state_vector = FALSE;
     }
  }
  else if ((strcmp (keyword, "POSITION") == 0) ||
           (strcmp (keyword, "LOCATION") == 0) )
  {
     if (TRACE) printf ("[AUVsocket POSITION]\n");

     state_variables_received = sscanf (command_received, "%s%lf%lf%lf",
                                         keyword,     &AUV_x_temp,
                                        &AUV_y_temp, &AUV_z_temp);

     if (state_variables_received == 4)  // transfer OK, keep new values
     {
       AUV_x                 = AUV_x_temp;
       AUV_y                 = AUV_y_temp;
       AUV_z                 = AUV_z_temp;
       return_state_vector = -4;
       printf ("%s\n", command_received);
     }
     else
     {
       printf ("a bad position was received & ignored:  %s",
                                                          command_received);
       return_state_vector = FALSE;
     }
  }
  else if ((strcmp (keyword, "ROTATION")     == 0) ||
```

```
            (strcmp (keyword, "ORIENTATION") == 0))
    {
        state_variables_received = sscanf (command_received, "%s%lf%lf%lf",
                                           keyword,        &AUV_phi_temp,
                                          &AUV_theta_temp, &AUV_psi_temp);

        if (state_variables_received == 4)  // transfer OK, keep new values
        {
          AUV_phi              = radians (AUV_phi_temp);
          AUV_theta            = radians (AUV_theta_temp);
          AUV_psi              = radians (AUV_psi_temp);
          return_state_vector = -4;
          printf ("%s\n", command_received);
        }
        else
        {
          printf ("a bad orientation was received & ignored:  %s",
                                                    command_received);
          return_state_vector = FALSE;
        }
    }
    else if (strncmp (command_received, "AUDIBLE", 7) == 0) /* enable sound */
    {
        printf ("[AUVsocket AUDIBLE]\n");

        return_state_vector = FALSE;
        audio_enabled       = TRUE;
        return (return_state_vector);  /* duplicate command copy not spoken */
    }
    else if (strncmp (command_received, "SILENT", 6) == 0) /* disable sound */
    {
        printf ("[AUVsocket SILENT]\n");

        return_state_vector = FALSE;
        audio_enabled       = FALSE;
    }

    if   ((strncmp (keyword, "AUV_STATE", 10) != 0) /* non-telemetry string */
       && (audio_enabled == TRUE))
    {
         /*****************************************************************/
         /* generate audio of non-telemetry line passed by execution level  */
         /* Don Brutzman  Naval Postgraduate School  brutzman@nps.navy.mil  */
         /*****************************************************************/

         start_index = 0;
         offset      = 0;
         strcpy (command_copy, "");

         /* leading blanks in query are stripped                          */
         for (index = 0; index <= strlen (command_received); index ++)
         {
              if  (command_received [index] == ' ') start_index = index + 1;
              else break;
         }
         /* clean up any extra stuff in the query string                  */
//         if  (command_received [strlen (command_received)] == '\n')
//       command_received [strlen (command_received)  ] = '}';
//       command_received [strlen (command_received)-1] = '{';

         int linelength = strlen (command_received);
         command_received [linelength] = (char) 0;

// printf (" command_received [linelength] = {%d}\n",
//          command_received [linelength]);

         for (index = start_index; index <= linelength; index ++)
```

```
{
    if      ((command_received [index]    == ' ') &&
             (command_received [index+1] == ' '))
              offset = offset - 1; // ignore multiple blanks
    else if ((command_received [index]    == ' ') &&
             (isdigit (command_received [index+1])))
          {
            command_copy [index - start_index + offset] = ',';
            // don't put minus before number, use comma instead
          }
    else if  (command_received [index] == ' ')
              command_copy [index - start_index + offset] = '-';
    else if  (command_received [index] == '\n')
          {
              command_copy [index - start_index + offset] =(char) 0;
              break;
          }
    else command_copy [index - start_index + offset] =
                                  tolower (command_received [index]);
}
           printf (" Say...%s\n",  command_received);
if (TRACE) printf ("[Say...%s]\n", command_copy);

/* build the remote query which will also save results as -object  */


strcpy (keyword, "speech/");
strcat (keyword, command_copy);
strcat (keyword, ".au");

checksoundfile = fopen (keyword, "r");

if (checksoundfile == NULL) /* file not previously retrieved-do so */
{
   if (TRUE)
   {
       printf ("[%s not found, making remote query to \n", keyword);
       printf (" the Say.. audio server in Netherlands]\n");
   }
   sprintf (www_execution_message_string,
     "www -o speech/%s.au http://www_tios.cs.utwente.nl/say/?%s",
                    command_copy,                            command_copy);
   if (TRACE) printf  ("[%s]\n", www_execution_message_string);
   system  (www_execution_message_string);
}
else
{
   if (TRACE)
       printf ("[%s was found locally, no remote query required]\n",
                 keyword);
   fclose (checksoundfile);
}
/* build string to play the audio file, then do so.               */
/* don't put sfplay in background or audio port may be unavailable */
/*       when the next audio message is sent                      */
sprintf (www_execution_message_string, "sfplay speech/%s.au",
                                                        command_copy);
if (TRACE) printf ("[%s]\n", www_execution_message_string);
system  (www_execution_message_string);

/*****************************************************************/
/* audio send complete *******************************************/
/*****************************************************************/
 }

}  /* end processing a message */
```

```
   return (return_state_vector);

} /* end read_from_execution_level_socket ();                                    */

/*****************************************************************************/
/*****************************************************************************/

void write_to_execution_level_socket ()  // using global variables state vector
{
   sprintf (command_sent,
" uvw_state %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f
%5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %5.3f
%5.3f %5.3f %5.3f %5.3f %5.3f %5.3f %d %5.3f %5.3f \n",
            AUV_time, AUV_x, AUV_y, AUV_z,
            normalize2 (degrees (AUV_phi)),
            normalize2 (degrees (AUV_theta)),
            normalize  (degrees (AUV_psi)),
            AUV_u, AUV_v, AUV_w,
            normalize2 (degrees (AUV_p)),
            normalize2 (degrees (AUV_q)),
            normalize2 (degrees (AUV_r)),
            AUV_x_dot,  AUV_y_dot,  AUV_z_dot,
            normalize2 (degrees (AUV_phi_dot)),
            normalize2 (degrees (AUV_theta_dot)),
            normalize2 (degrees (AUV_psi_dot)),
            normalize2 (degrees (AUV_delta_rudder)),
            normalize2 (degrees (AUV_delta_planes)),
            AUV_port_rpm,         AUV_stbd_rpm,
            AUV_bow_vertical,     AUV_stern_vertical,
            AUV_bow_lateral,      AUV_stern_lateral,
            AUV_ST1000_bearing,   AUV_ST1000_range,     AUV_ST1000_strength,
            AUV_ST725_bearing,    AUV_ST725_range,      AUV_ST725_strength
            );

    if (TRACE) cout << command_sent << endl;

   /*************************************************************************/
   /* Sender block                                                          */
   /*************************************************************************/

   bytes_received = strlen (command_sent);

   if (bytes_received < 0)  /* copy failure                                 */
   {
       cout << "write_to_execution_level_socket () ";
       cout << "copy from command_received unsuccessful" << endl;
       shutdown_socket (0);
       return;
   }

   else if (bytes_received > socket_length)
        {
           cout << "write_to_execution_level_socket () "
                << "send_telemetry_to_server error: " << endl;
           cout << "bytes_received too big for packet socket_length"
                << "             [bytes_received=" << bytes_received
                << "] > [socket_length=" << socket_length << "]; "
                << "string truncated" << endl;
        }

   bytes_left        = socket_length;
   bytes_written     = 0;
   ptr_index         = command_sent;

   while ((bytes_left > 0) && (bytes_written >= 0))  /* write loop ***********/
   {
       bytes_sent = write (socket_stream, ptr_index, bytes_left);
```

```
        if      (bytes_sent <  0) bytes_written = bytes_sent;
        else if (bytes_sent >  0)
             {
                  bytes_left     -= bytes_sent;
                  bytes_written  += bytes_sent;
                  ptr_index      += bytes_sent;
             }
        if (TRACE && (bytes_written != socket_length))
        {
            cout << "write_to_execution_level_socket () loop  bytes sent = "
                 << bytes_sent << endl;
        }
    }
    if   (bytes_written < 0)
    {
          cout << "write_to_execution_level_socket () send failed, "
               << bytes_written << " bytes_written" << endl;
    }
    else if (TRACE)
         {
           cout << "write_to_execution_level_socket () total bytes sent = "
                << bytes_written << endl;
         }

    return;

}  /* end write_to_execution_level_socket ();
*/

/****************************************************************************/
/****************************************************************************/

int  shutdown_socket (int signal_thrown)                     /* Shutdown block */

{
    int close_result;

    if (TRACE)
        cout << "shutdown_socket (" << signal_thrown
             << ") shutdown in progress ..." << endl;

    shutdown_signal_received = TRUE;

    /* No need to send a message to other side that bridge is going down,      */
    /*    since SIGPIPE signal trigger may cause shutdown on other side        */

    close_result = close (socket_stream);

    if (close_result == -1)
        cout << "shutdown_socket () close (socket_stream) failed" << endl;

    if (TRUE) cout << "shutdown_socket () complete" << endl;

    return (close_result);

} /* end shutdown_socket () */

/****************************************************************************/
/****************************************************************************/

double normalize (double degs)      /* degrees input*/
{
    double result = degs;

    while (result <    0.0) result += 360.0;
    while (result >= 360.0) result -= 360.0;
```

```
   return result;
}
/*************************************************************************/
/*************************************************************************/

double normalize2 (double degs)     /* degrees input*/
{
   double result = degs;

   while (result <= -180.0) result += 360.0;
   while (result >   180.0) result -= 360.0;

   return result;
}
/*************************************************************************/
/*************************************************************************/

void AUV_global_initialization ()
{
// Variable initialization performed separately to avoid compiler warnings - -

              AUV_time           = 0.0; // mission time
              AUV_delta_rudder   = 0.0; // positive is bow rudder _______
              AUV_delta_planes   = 0.0; // positive is bow planes _______
              AUV_port_rpm       = 0.0; // propellor revolutions per minute
              AUV_stbd_rpm       = 0.0; // propellor revolutions per minute

              AUV_bow_vertical   = 0.0; // thruster volts 24V = 3820 rpm no load
              AUV_stern_vertical = 0.0; // thruster volts 24V = 3820 rpm no load
              AUV_bow_lateral    = 0.0; // thruster volts 24V = 3820 rpm no load
              AUV_stern_lateral  = 0.0; // thruster volts 24V = 3820 rpm no load

              AUV_depth_cell     = 0.0; // pressure sensor, units are ________
              AUV_heading        = 0.0; // gyrocompass in degrees
              AUV_roll_angle     = 0.0; // matches intertial sensor onboard AUV
              AUV_pitch_angle    = 0.0; // matches intertial sensor onboard AUV
              AUV_roll_rate      = 0.0; // matches intertial sensor onboard AUV
              AUV_pitch_rate     = 0.0; // matches intertial sensor onboard AUV
              AUV_yaw_rate       = 0.0; // matches intertial sensor onboard AUV

              AUV_hour           = 0;   // internal AUV OS-9 system time, unused
              AUV_minute         = 0;
              AUV_second         = 0;


// Hydrodynamics-model-provided state variables - - - - - - - - - - - - - - -

              AUV_x              = 0.0; // x     position in world coordinates
              AUV_y              = 0.0; // y     position in world coordinates
              AUV_z              = 0.0; // z     position in world coordinates
              AUV_phi            = 0.0; // roll  posture in world coordinates
              AUV_theta          = 0.0; // pitch posture in world coordinates
              AUV_psi            = 0.0; // yaw   posture in world coordinates

              AUV_x_dot          = 0.0; //      Euler velocity along North-axis
              AUV_y_dot          = 0.0; //      Euler velocity along  East-axis
              AUV_z_dot          = 0.0; //      Euler velocity along Depth-axis
              AUV_phi_dot        = 0.0; // Euler rotation rate about North-axis
              AUV_theta_dot      = 0.0; // Euler rotation rate about  East-axis
              AUV_psi_dot        = 0.0; // Euler rotation rate about Depth-axis

              AUV_u              = 0.0; // surge  linear velocity along x-axis
              AUV_v              = 0.0; // sway   linear velocity along y-axis
              AUV_w              = 0.0; // heave  linear velocity along x-axis
              AUV_p              = 0.0; // roll  angular velocity about x-axis
              AUV_q              = 0.0; // pitch angular velocity about y-axis
              AUV_r              = 0.0; // yaw   angular velocity about z-axis
```

```
            AUV_u_dot           = 0.0; //    linear acceleration along x-axis
            AUV_v_dot           = 0.0; //    linear acceleration along y-axis
            AUV_w_dot           = 0.0; //    linear acceleration along x-axis
            AUV_p_dot           = 0.0; //   angular acceleration about x-axis
            AUV_q_dot           = 0.0; //   angular acceleration about y-axis
            AUV_r_dot           = 0.0; //   angular acceleration about z-axis

            AUV_oceancurrent_u = 0.0; //  Ocean current rate along North-axis
            AUV_oceancurrent_v = 0.0; //  Ocean current rate along  East-axis
            AUV_oceancurrent_w = 0.0; //  Ocean current rate along Depth-axis


// Prior time loop saved variables - - - - - - - - - - - - - - - - - - - - - - - -

            AUV_time_prior      = 0.0; // mission time

            AUV_x_prior         = 0.0; // x    position in world coordinates
            AUV_y_prior         = 0.0; // y    position in world coordinates
            AUV_z_prior         = 0.0; // z    position in world coordinates

            AUV_phi_prior       = 0.0; // roll  posture in world coordinates
            AUV_theta_prior     = 0.0; // pitch posture in world coordinates
            AUV_psi_prior       = 0.0; // yaw   posture in world coordinates

// Sonar- & terrain-model-provided state variables - - - - - - - - - - - - - - -

            AUV_ST1000_bearing = 0.0; // ST_1000 conical pencil beam bearing
            AUV_ST1000_range   = 0.0; // ST_1000 conical pencil beam range
            AUV_ST1000_strength= 0.0; // ST_1000 conical pencil beam strength


            AUV_ST725_bearing  = 0.0; // ST_725  1 x 24  sector beam bearing
            AUV_ST725_range    = 0.0; // ST_725  1 x 24  sector beam range
            AUV_ST725_strength = 0.0; // ST_725  1 x 24  sector beam strength

;


#endif  /* AUVSOCKET_C */
```

G. *DISNetworkedRigidBody.C* DIS Network Connections for a Rigid Body

```
////////////////////////////////////////////////////////////////////////////////
/*
 Program:          DISNetworkedRigidBody.C

 Description:      DIS network interface for RigidBody model

 Author:           Don Brutzman

 Revised:          28 October 94

 System:           Irix 5.2

 Compiler:         ANSI C++

 Compilation:      irix> make dynamics
                   irix> CC DISNetworkedRigidBody.C -lm -c -g +w

 Original bases:   os9server.c, dynamics.c, DIS library test_it.c

 References:   (1) IEEE Protocols for Distributed Interactive Simulation (DIS)
                   Applications version 2.0, Institute for Simulation and
                   Training, Universit of Central Florida, Orlando Florida,
                   28 May 1993.

               (2) Macedonia, Michael, Zeswitz, Steven, and Locke, John,
                   Distributed Interactive Simulation (DIS) multicast
                   version 2.0.3, Naval Postgraduate School, February 94.

               (3) Zeswitz, Steven, "NPSNET: Integration of Distributed
                   Interactive Simulation (DIS) Protocol for Communication
                   Architecture and Information Interchange." master's thesis,
                   Naval Postgraduate School, Monterey California, 28 May 1993.

Units:             Virtual world:  ft    ft/sec   radians   radians/sec
                   DIS PDUs:       m      m/sec   degrees   degrees/sec

Future work:


Advisors:          Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

*/
////////////////////////////////////////////////////////////////////////////////

#ifndef DISNETWORKEDRIGIDBODY_C
#define DISNETWORKEDRIGIDBODY_C // prevent errors if multiple #includes present

#define  METERS_PER_FT  0.3048
#define  FT_PER_METERS  3.2808

#include "RigidBody.C"
#include "AUVsocket.C"
#include <string.h>
#include <bstring.h>
#include <time.h>

// DIS includes.  See Makefile for other DIS #include files; they must match.

#include "../DIS.mcast/h/disdefs.h"
#include "../DIS.mcast/h/appearance.h"

////////////////////////////////////////////////////////////////////////////////
```

```
// DIS library function prototypes do not include parameter prototypes, thus
//     needed ones are included here.  Function definitions are located in
//     /n/elsie/work3/macedoni/net/mcast/network/src

///////////////////////////////////////////////////////////////////////////////////

class DISNetworkedRigidBody : public RigidBody
{
private:

// member data fields

   double            time_of_last_PDU;

   int               DIS_port_open;

   EntityID          UUV_DIS_id;

   EntityType        UUV_DIS_type;

   EntityStatePDU  * UUV_DIS_pdu;

   clock_t           current_clock,    // clock tick counter for current loop
                     initialclock,
                     previous_clock,
                     delta_clock;
   double            delta_time;

   int               PDU_skip_interval;

   // Multicast Defaults from
   // /n/elsie/work3/macedoni/net/mcast/network/utils/planes/planes.cc

   char   MULTICAST_PORT   [6];
   char   MULTICAST_GROUP [20];
   char   port             [6];         // pointer to MULTICAST_PORT
   char   group           [20];         // MULTICAST_GROUP
   u_char ttl;                          // time-to-live

public:

// member constructor and destructor functions

   DISNetworkedRigidBody                 ();

  ~DISNetworkedRigidBody                 () { /* null body */ }

// operators

   friend ostream& operator <<           (ostream& out,
                                          DISNetworkedRigidBody& nrb);

// inspection methods

   void      print_networkedrigidbody    () const;

   double    time_of_last_PDU_value      () const;

   int       DIS_port_open_value         () const;

   int       PDU_skip_interval_value     () const;

// modifying methods

   void      set_PDU_skip_interval       (const int new_value);

   int       DIS_net_open                ();
```

```
   int        DIS_net_write                 ();
   void       DIS_net_close                 ();
   void       DISNetworkedRigidBody_initialize ();
   void       set_time_of_last_PDU          (const double new_time_of_last_PDU);
   void       set_ttl                       (int    new_ttl);
   void       set_group                     (char * new_group);
   void       set_port                      (char * new_port);
};
/////////////////////////////////////////////////////////////////////////////////

//-----------------------------------------------------------------------------//
// constructor methods
//-----------------------------------------------------------------------------//

DISNetworkedRigidBody:: DISNetworkedRigidBody () // default constructor
{
   PDU_skip_interval = 1;              // 1 tenth(s) of a second between PDUs

   RigidBody_initialize ();                // inherited method
   time_of_last_PDU = 0.0;
   DIS_port_open = FALSE;

   // initialize clock variables
   initialclock      = clock ();
   previous_clock    = clock ();

   // Multicast Defaults from
   // /n/elsie/work3/macedoni/net/mcast/network/utils/planes/planes.cc

   bzero  (MULTICAST_PORT,    6);
   bzero  (MULTICAST_GROUP, 20);
   strcpy (MULTICAST_PORT,   "3111");          // 3111 npsnet 'standard' address
   strcpy (MULTICAST_GROUP, "224.2.121.93");

   bzero  (port,   6);
   bzero  (group, 20);
   strcpy (port,  MULTICAST_PORT);
   strcpy (group, MULTICAST_GROUP);

   ttl    = 16;              // multicast ttl=16 stays inside NPS
}
//-----------------------------------------------------------------------------//
// operators
//-----------------------------------------------------------------------------//

ostream& operator << (ostream& out, DISNetworkedRigidBody& nrb)
{
   nrb.print_rigidbody ();
   out   << "time_of_last_PDU = "  << nrb.time_of_last_PDU << endl;
   return (out);
}
//-----------------------------------------------------------------------------//
// inspection methods
//-----------------------------------------------------------------------------//

void DISNetworkedRigidBody:: print_networkedrigidbody ()
const
{
```

```
   print_rigidbody ();
   cout << "time_of_last_PDU   = "  << time_of_last_PDU << endl;
}
//------------------------------------------------------------------------------//

double DISNetworkedRigidBody:: time_of_last_PDU_value ()
const
{
   return time_of_last_PDU;
}
//------------------------------------------------------------------------------//

int DISNetworkedRigidBody:: DIS_port_open_value ()
const
{
   return DIS_port_open;
}
//------------------------------------------------------------------------------//

int DISNetworkedRigidBody:: PDU_skip_interval_value ()
const
{
   return PDU_skip_interval;
}
//------------------------------------------------------------------------------//
// modifying methods
//------------------------------------------------------------------------------//

void DISNetworkedRigidBody:: set_PDU_skip_interval (const int new_value)
{
   if   (PDU_skip_interval >= 1) PDU_skip_interval = new_value;
   else
   {
       PDU_skip_interval = 10;
       cout << "PDU_skip_interval must be >= 1, and has been set to 1." << endl;
   }

   return;
}
//------------------------------------------------------------------------------//

int DISNetworkedRigidBody:: DIS_net_open ()   // Ref: macedonia include files
{
   int    exercise_id       = -1;
   int    coordinate_system = 0;  // 0 = flat world, 1 = round world
   char * utm_file          = "";

   int result = FALSE;

   if (DIS_port_open == FALSE)
   {
       result = net_open (port, group, ttl,
                          exercise_id, coordinate_system, utm_file);

       // result = net_open (port, group, ttl); // old version

       if  (result == FALSE)
       {
            DIS_port_open = FALSE;
            cout << "DIS_net_open_select () failed" << endl;
       }
       else
       {
            DIS_port_open = TRUE;
            // atexit (net_close); // ensure port is reclosed on exit.
        }
  }
```

```
   else
   {
       cout << "DIS_port_open == TRUE so DIS_net_open () not re-attempted."
            << endl;
       result = TRUE;
   }
   return result;
}
//--------------------------------------------------------------------------//

int DISNetworkedRigidBody:: DIS_net_write ()
{
   current_clock      = clock ();
   delta_clock        = current_clock - previous_clock;
   delta_time         = delta_clock / CLOCKS_PER_SEC;

   if (TRACE) cout << "DISNetworkedRigidBody::DIS_net_write:AUV_time = "
                   << AUV_time;

   // do not write faster than one DIS PDU per PDU_skip_interval tenth-seconds
   //    of AUV_time
   if ((int) (AUV_time * 10.0) % PDU_skip_interval != 0)
   {
        if (TRACE) cout << ", returning." << endl;
        return (TRUE);
   }
   else if (TRACE) cout << ", sending DIS PDU." << endl;

   previous_clock = current_clock;

   // see test_it.c ('client' program) in DIS library for examples:
   //      /n/elsie/work3/macedonia/net/mcast/network/src/test_it.c

   UUV_DIS_id.address.site  = SITE_ID_NPS;  // DIS standard para 5.3.8.1.1

   UUV_DIS_id.address.host  = (unsigned short) 1;
   UUV_DIS_id.entity        = (unsigned short) 1;

   // force the #define values to the right types
   UUV_DIS_type.entity_kind = (unsigned char)  EntityKind_Platform;
   UUV_DIS_type.domain      = (unsigned char)  Domain_Subsurface;
   UUV_DIS_type.country     = (unsigned short) USA;
// not yet defined in DIS header file
// UUV_DIS_type.category    = (unsigned char)  Category_ResearchMiscSub;
   UUV_DIS_type.subcategory = (unsigned char)  0;
   UUV_DIS_type.specific    = (unsigned char)  0;

   UUV_DIS_pdu = mallocEntityStatePDU();

   // fill in the parameters of an entity state PDU (most are listed in pdu.h)
   //          this assumes there are no articulated parameters (add later) <<<<<

   // Note all units converted to SI when building PDU

   // DIS ID and Type
   UUV_DIS_pdu->entity_id              = UUV_DIS_id;
   UUV_DIS_pdu->entity_type            = UUV_DIS_type;
   UUV_DIS_pdu->force_id               = ForceID_White;
   UUV_DIS_pdu->alt_entity_type.extra = (Extra) 0x00;  // unused extra parameter

   // Posture
   UUV_DIS_pdu->entity_location.x        = AUV_x * METERS_PER_FT;
   UUV_DIS_pdu->entity_location.y        = AUV_y * METERS_PER_FT;
   UUV_DIS_pdu->entity_location.z        = AUV_z * METERS_PER_FT;
   UUV_DIS_pdu->entity_orientation.psi   = (int) degrees (AUV_psi)   % 360;
   UUV_DIS_pdu->entity_orientation.theta = (int) degrees (AUV_theta) % 360;
   UUV_DIS_pdu->entity_orientation.phi   = (int) degrees (AUV_phi)   % 360;
```

```
    // Linear and angular velocities in body coordinates/meters by DIS standard
    UUV_DIS_pdu->entity_velocity.x          = AUV_x_dot * METERS_PER_FT;
    UUV_DIS_pdu->entity_velocity.y          = AUV_y_dot * METERS_PER_FT;
    UUV_DIS_pdu->entity_velocity.z          = AUV_z_dot * METERS_PER_FT;

    UUV_DIS_pdu->dead_reckon_params.angular_velocity [0] =
                                        (int) degrees (AUV_phi_dot)   % 360;

    UUV_DIS_pdu->dead_reckon_params.angular_velocity [1] =
                                        (int) degrees (AUV_theta_dot) % 360;

    UUV_DIS_pdu->dead_reckon_params.angular_velocity [2] =
                                        (int) degrees (AUV_psi_dot)   % 360;

    // no explicit world coordinate accelerations are provided by dynamics model
    UUV_DIS_pdu->dead_reckon_params.linear_accel [0]     = 0.0;
    UUV_DIS_pdu->dead_reckon_params.linear_accel [1]     = 0.0;
    UUV_DIS_pdu->dead_reckon_params.linear_accel [2]     = 0.0;

//  debug code:
//  UUV_DIS_pdu->entity_velocity.x                       = 0.0;
//  UUV_DIS_pdu->entity_velocity.y                       = 0.0;
//  UUV_DIS_pdu->entity_velocity.z                       = 0.0;
//  UUV_DIS_pdu->dead_reckon_params.angular_velocity [0] = 0;
//  UUV_DIS_pdu->dead_reckon_params.angular_velocity [1] = 0;
//  UUV_DIS_pdu->dead_reckon_params.angular_velocity [2] = 0;

if (TRUE)
    {
      cout << endl;
      cout << "DIS_net_write:  time " << AUV_time << ":  ";
      cout << "[";
      cout << UUV_DIS_pdu->entity_location.x         << ", ";
      cout << UUV_DIS_pdu->entity_location.y         << ", ";
      cout << UUV_DIS_pdu->entity_location.z         << ", ";
      cout << UUV_DIS_pdu->entity_orientation.phi    << ", ";
      cout << UUV_DIS_pdu->entity_orientation.theta  << ", ";
      cout << UUV_DIS_pdu->entity_orientation.psi    << "]" << endl;
    }
    if (TRACE)
    {
      cout << "World coordinate velocities:    ";
      cout << "[";
      cout << UUV_DIS_pdu->entity_velocity.x         << ", ";
      cout << UUV_DIS_pdu->entity_velocity.y         << ", ";
      cout << UUV_DIS_pdu->entity_velocity.z         << ", ";
      cout << UUV_DIS_pdu->dead_reckon_params.angular_velocity [0] << ", ";
      cout << UUV_DIS_pdu->dead_reckon_params.angular_velocity [1] << ", ";
      cout << UUV_DIS_pdu->dead_reckon_params.angular_velocity [2] << "]"
           << endl;
    }

    if (TRACE)
    {
      cout << "World coordinate accelerations: ";
      cout << "[";
      cout << UUV_DIS_pdu->dead_reckon_params.linear_accel [0] << ", ";
      cout << UUV_DIS_pdu->dead_reckon_params.linear_accel [1] << ", ";
      cout << UUV_DIS_pdu->dead_reckon_params.linear_accel [2] << "]" << endl;
    }

    // what we look like
    UUV_DIS_pdu->entity_appearance = AppearanceSubSurf_SmallWake;
    UUV_DIS_pdu->entity_marking.character_set = CharSet_ASCII;
    strncpy ((char *) UUV_DIS_pdu->entity_marking.markings, "NPS AUV    ",
                                                              MARKINGS_LEN);
```

```
ArticulatParamsNode * APNptr;

// articulated parameters:  rudders, planes,  test with execution level rpm,
stbd rpm

// note that the function specification for attachArticulatParamsNode
// needed to be corrected to acceot (char *) in file disdefs.h

if (APNptr = attachArticulatParamsNode ((char *) UUV_DIS_pdu,
                                         EntityStatePDU_Type))
{
   // Successful attach, fill in data.  See DIS standard paragraph 5.3.3
   APNptr->articulat_params.change = TRUE; // always active
   APNptr->articulat_params.ID     = 0;    // articulated parameter #
   APNptr->articulat_params.parameter_value [0]  // auv time, even seconds
                           = (unsigned short) ((int) AUV_time % 600);
   APNptr->articulat_params.parameter_value [1] // auv time tenths of seconds
                           = (unsigned short) (     AUV_time * 10) % 10;
   APNptr->articulat_params.parameter_value [2]
                           = (signed short) ((int) degrees (AUV_delta_rudder)
                                                   % 360);
   APNptr->articulat_params.parameter_value [3]
                           = (signed short) ((int) degrees (AUV_delta_planes)
                                                   % 360);
   APNptr->articulat_params.parameter_value [4]
                           = (signed short) ((int)       AUV_port_rpm  / 10);
   APNptr->articulat_params.parameter_value [5]
                           = (signed short) ((int) fabs (AUV_port_rpm) % 10);
   APNptr->articulat_params.parameter_value [6]
                           = (signed short) ((int)       AUV_stbd_rpm  / 10);
   APNptr->articulat_params.parameter_value [7]
                           = (signed short) ((int) fabs (AUV_stbd_rpm) % 10);
}
else cout << "[error attaching articulated parameters node 0:  time, "
          << "rudder, planes, rpm]" << endl;

// articulated parameters:  Thruster parameters
if (APNptr = attachArticulatParamsNode ((char *) UUV_DIS_pdu,
                                         EntityStatePDU_Type))
{
   // Successful attach, fill in data.  See DIS standard paragraph 5.3.3
   APNptr->articulat_params.change = TRUE; // always active
   APNptr->articulat_params.ID     = 1;    // articulated parameter #
   APNptr->articulat_params.parameter_value [0] = (int) AUV_bow_vertical;
   APNptr->articulat_params.parameter_value [1] = (int) AUV_stern_vertical;
   APNptr->articulat_params.parameter_value [2] = (int) AUV_bow_lateral;
   APNptr->articulat_params.parameter_value [3] = (int) AUV_stern_lateral;
   APNptr->articulat_params.parameter_value [4] = 0;
   APNptr->articulat_params.parameter_value [5] = 0;
   APNptr->articulat_params.parameter_value [6] = 0;
   APNptr->articulat_params.parameter_value [7] = 0;
}
else cout << "[error attaching articulated parameters node 1: thrusters]"
          << endl;

// articulated parameters:  Sonar parameters
if (APNptr = attachArticulatParamsNode ((char *) UUV_DIS_pdu,
                                         EntityStatePDU_Type))
{
   // Successful attach, fill in data.  See DIS standard paragraph 5.3.3
   APNptr->articulat_params.change = TRUE; // always active
   APNptr->articulat_params.ID     = 1;    // articulated parameter #
   APNptr->articulat_params.parameter_value [0]
                           = (signed short) ((int) AUV_ST1000_bearing) / 10;
   APNptr->articulat_params.parameter_value [1]
                           = (signed short) ((int) AUV_ST1000_bearing) % 10;
   APNptr->articulat_params.parameter_value [2]
```

```
                              = (signed short)
((int)(AUV_ST1000_range*4.0+0.5));
    APNptr->articulat_params.parameter_value [3]
                              = (signed short) ((int) AUV_ST1000_strength);
    APNptr->articulat_params.parameter_value [4]
                              = (signed short) ((int) AUV_ST725_bearing) / 10;
    APNptr->articulat_params.parameter_value [5]
                              = (signed short) ((int) AUV_ST725_bearing) % 10;
    APNptr->articulat_params.parameter_value [6]
                              = (signed short) ((int)(AUV_ST725_range*4.0+0.5));
    APNptr->articulat_params.parameter_value [7]
                              = (signed short) ((int) AUV_ST725_strength);
  }
  else cout << "[error attaching articulated parameters node 2:  sonar]"
            << endl;

  // Identify dead reckoning algorithm:  world coordinates, accelerations zero
  //                                     (Algorithm 3)
  UUV_DIS_pdu->dead_reckon_params.algorithm = DRAlgo_DRM_RPW;

  // send out the multicast PDU
  int  result = net_write ((char *) UUV_DIS_pdu, EntityStatePDU_Type);

  if  (result == FALSE)
  {
      cout << "DIS_net_write () failed" << endl;
  }
  else if (TRACE) cout << "DIS_net_write () successful, returning" << endl;

  freeEntityStatePDU (UUV_DIS_pdu);  // essential to prevent memory leak
                                     // articulated parameters are also freed

  return result;  // end of DIS_net_write ()
}
//-----------------------------------------------------------------------//

void DISNetworkedRigidBody:: DIS_net_close ()
{
  DIS_port_open = FALSE;
  net_close ();
}
//-----------------------------------------------------------------------//

void DISNetworkedRigidBody:: DISNetworkedRigidBody_initialize ()
{
  RigidBody_initialize ();                 // inherited method
  time_of_last_PDU = 0.0;
}
//-----------------------------------------------------------------------//

void DISNetworkedRigidBody:: set_time_of_last_PDU
                                 (const double new_time_of_last_PDU)
{
  time_of_last_PDU = new_time_of_last_PDU;
}
//-----------------------------------------------------------------------//

void DISNetworkedRigidBody:: set_ttl  (int    new_ttl)
{
  cout << "DISNetworkedRigidBody::set_ttl:   old ttl   = " << (int) ttl;

  if (new_ttl > 16)
  {
     cout << endl;
     cout << "A large ttl value may impact people outside your network "
          << endl;
     cout << "and even around the world!" << endl;
```

```
    cout << "Please confirm by entering the new time to live (ttl) "
         << "value (" << new_ttl << ") yourself:  ";
    cin  >> new_ttl;
    cout << endl;

   if ((new_ttl < 1) || (new_ttl > 255))
   {
    cout << endl;
    cout << "Time to live (ttl) value out of range (1..255), ignored."
         << endl;
    return;
   }

   ttl = (u_char) new_ttl;

   cout << endl;
   cout << "                                       new ttl   = "
        << (int) ttl << endl;
}
//--------------------------------------------------------------------------//

void DISNetworkedRigidBody:: set_group  (char * new_group)
{
   cout << "DISNetworkedRigidBody::set_group: old group = " << group;

   bzero  (group,  20);
   strcpy (group, new_group);

   cout << endl;
   cout << "                                       new group = " << group << endl;
}
//--------------------------------------------------------------------------//

void DISNetworkedRigidBody:: set_port  (char * new_port)
{
   cout << "DISNetworkedRigidBody::set_port:  old port  = " << port;

   bzero  (port,   6);
   strcpy (port, new_port);

   cout << endl;
   cout << "                                       new port  = " << port << endl;
}
//--------------------------------------------------------------------------//

#endif // DISNETWORKEDRIGIDBODY_C
```

H. *RigidBody.C* Rigid Body

```
/////////////////////////////////////////////////////////////////////////////////
/*
 Program:          RigidBody.C

 Author:           Don Brutzman

 Revised:          18 October 94

 System:           Irix 5.2

 Compiler:         ANSI C++

 Compilation:      irix> make dynatest
                   irix> CC RigidBody.C -lm -c -g +w

                    -c == Produce binaries only, suppressing the link phase.
                    +w == Warn about all questionable constructs.

 Description:      Rigid body class specification.
                   Note: world coordinate system, NOT body coordinate system,
                         thus Euler angles are used.

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 References:       Fu, K.S., Gonzalez, R.C., and Lee, C.S.G.,
                   _Robotics:  Control, Sensing, Vision and Intelligence_,
                   McGraw-Hill Inc., NY, 1987.

                   Cooke, J.C., Zyda, M.J., Pratt, D.R., and McGhee, R.B.,
                   "NPSNET:  Flight Simulation Dynamic Modeling using
                   Quaternions," _PRESENCE_, vol. 1 no. 4, Fall 1992,
                   pp. 404-420.

                   Cooke, Joseph C., "NPSNET:  Flight Simulation Dynamic
                   Modeling using Quaternions," masters thesis, Naval
                   Postgraduate School, December 1993.

*/
/////////////////////////////////////////////////////////////////////////////////

#ifndef RIGIDBODY_C
#define RIGIDBODY_C     // prevent errors if multiple #includes present

#include "Hmatrix.C"

class RigidBody
{
private:

// member data fields

   double    time_of_posture;

   double       x_dot;          // linear  velocity along x axis
   double       y_dot;          // linear  velocity along y axis
   double       z_dot;          // linear  velocity along z axis
   double     phi_dot;          // angular velocity about x axis
   double   theta_dot;          // angular velocity about y axis
   double     psi_dot;          // angular velocity about z axis

public:
```

```
    Hmatrix    hmatrix;           // need to access print & set routines

// constructors and destructor

    RigidBody                     ();
    RigidBody                     (const Hmatrix initialhmatrix);

    RigidBody                     (const Hmatrix initialhmatrix, const double t);

    RigidBody                     (const double x,
                                   const double y,
                                   const double z,
                                   const double phi,
                                   const double theta,
                                   const double psi,
                                   const double t);

   ~RigidBody                     ()                { /* null body */ }

// operators

    RigidBody&    operator =      (const RigidBody  &rb_rhs);      // assignment

    friend ostream & operator << (ostream& out, RigidBody& rb);

// inspection methods

    void        print_rigidbody         () const;

    Hmatrix     hmatrix_value           () const;
    double      time_of_posture_value   () const;

    double          x_value             () const;
    double          y_value             () const;
    double          z_value             () const;
    double        phi_value             () const;
    double      theta_value             () const;
    double        psi_value             () const;

    double          x_dot_value         () const;
    double          y_dot_value         () const;
    double          z_dot_value         () const;
    double        phi_dot_value         () const;
    double      theta_dot_value         () const;
    double        psi_dot_value         () const;

// modifying methods

    void        RigidBody_initialize    ();

    void        set_velocities          (double new_x_dot,
                                         double new_y_dot,
                                         double new_z_dot,
                                         double new_phi_dot,
                                         double new_theta_dot,
                                         double new_psi_dot);

    void        set_linear_velocities   (double new_x_dot,
                                         double new_y_dot,
                                         double new_z_dot);

    void        set_angular_velocities  (double new_phi_dot,
                                         double new_theta_dot,
                                         double new_psi_dot);

    void        set_time_of_posture     (double new_time_of_posture);
```

```
   void        update_Hmatrix         (double delta_t);

};

//////////////////////////////////////////////////////////////////////////////

//-------------------------------------------------------------------------//
// constructor methods
//-------------------------------------------------------------------------//

RigidBody:: RigidBody () // default constructor
{
   hmatrix.set_identity ();
   time_of_posture = 0.0;
       x_dot        = 0.0;
       y_dot        = 0.0;
       z_dot        = 0.0;
     phi_dot        = 0.0;
   theta_dot        = 0.0;
     psi_dot        = 0.0;
}
//-------------------------------------------------------------------------//

RigidBody:: RigidBody (const Hmatrix initialhmatrix)
{
   hmatrix = Hmatrix (initialhmatrix);
   time_of_posture = 0.0;
       x_dot        = 0.0;
       y_dot        = 0.0;
       z_dot        = 0.0;
     phi_dot        = 0.0;
   theta_dot        = 0.0;
     psi_dot        = 0.0;
}
//-------------------------------------------------------------------------//

RigidBody:: RigidBody (const Hmatrix initialhmatrix, const double t)
{
   hmatrix = Hmatrix (initialhmatrix);
   time_of_posture = t;
       x_dot        = 0.0;
       y_dot        = 0.0;
       z_dot        = 0.0;
     phi_dot        = 0.0;
   theta_dot        = 0.0;
     psi_dot        = 0.0;
}
//-------------------------------------------------------------------------//

RigidBody:: RigidBody (const double x,
                       const double y,
                       const double z,
                       const double phi,
                       const double theta,
                       const double psi,
                       const double t)
{
   hmatrix         = Hmatrix (phi, theta, psi, x, y, z);
   time_of_posture = t;
       x_dot       = 0.0;
       y_dot       = 0.0;
       z_dot       = 0.0;
     phi_dot       = 0.0;
   theta_dot       = 0.0;
     psi_dot       = 0.0;
}
```

```
//------------------------------------------------------------------------//

RigidBody& RigidBody:: operator = (const RigidBody  &rb_rhs)     // assignment
{
   hmatrix          = rb_rhs.hmatrix;
   time_of_posture = rb_rhs.time_of_posture;
       x_dot        = rb_rhs.x_dot;
       y_dot        = rb_rhs.y_dot;
       z_dot        = rb_rhs.z_dot;
     phi_dot        = rb_rhs.phi_dot;
   theta_dot        = rb_rhs.theta_dot;
     psi_dot        = rb_rhs.psi_dot;
   return *this;
}
//------------------------------------------------------------------------//


ostream& operator << (ostream& out, RigidBody& rb)
{
   rb.hmatrix.print_hmatrix ();
   out << "time_of_posture    = "  << rb.time_of_posture << endl;
   out << "< x_dot, y_dot, z_dot, phi_dot, theta_dot, psi_dot> = <"
       << rb.x_dot    << ", " << rb.y_dot      << ", " << rb.z_dot    << ", "
       << rb.phi_dot << ", " << rb.theta_dot << ", " << rb.psi_dot << ">"
       << endl;
   return out;
}
//------------------------------------------------------------------------//
// inspection methods
//------------------------------------------------------------------------//

void RigidBody:: print_rigidbody ()
const
{
   hmatrix.print_hmatrix ();;
   cout << "time_of_posture    = "  << time_of_posture << endl;
   cout << "< x_dot, y_dot, z_dot, phi_dot, theta_dot, psi_dot> = <"
        <<    x_dot << ", " <<      y_dot << ", " <<    z_dot << ", "
        <<  phi_dot << ", " << theta_dot << ", " << psi_dot << ">" << endl;
}
//------------------------------------------------------------------------//

Hmatrix RigidBody:: hmatrix_value ()    // note this returns an Hmatrix object,
const                                   //      not a [4][4] array
{
   return hmatrix;
}
//------------------------------------------------------------------------//

double RigidBody:: time_of_posture_value ()
const
{
   return time_of_posture;
}
//------------------------------------------------------------------------//

double RigidBody:: x_value ()
const
{
   return hmatrix.x_value ();
}
//------------------------------------------------------------------------//

double RigidBody:: y_value ()
const
{
   return hmatrix.y_value ();
```

```
}
//--------------------------------------------------------------------------//

double RigidBody:: z_value ()
const
{
   return hmatrix.z_value ();
}
//--------------------------------------------------------------------------//

double RigidBody:: phi_value ()
const
{
   return hmatrix.phi_value ();
}
//--------------------------------------------------------------------------//

double RigidBody:: theta_value ()
const
{
   return hmatrix.theta_value ();
}
//--------------------------------------------------------------------------//

double RigidBody:: psi_value ()
const
{
   return hmatrix.psi_value ();
}
//--------------------------------------------------------------------------//

double RigidBody:: x_dot_value ()
const
{
   return x_dot;
}
//--------------------------------------------------------------------------//

double RigidBody:: y_dot_value ()
const
{
   return y_dot;
}
//--------------------------------------------------------------------------//

double RigidBody:: z_dot_value ()
const
{
   return z_dot;
}
//--------------------------------------------------------------------------//

double RigidBody:: phi_dot_value ()
const
{
   return phi_dot;
}
//--------------------------------------------------------------------------//

double RigidBody:: theta_dot_value ()
const
{
   return theta_dot;
}
//--------------------------------------------------------------------------//

double RigidBody:: psi_dot_value ()
```

```
const
{
   return psi_dot;
}
//---------------------------------------------------------------------------//
// modifying methods
//---------------------------------------------------------------------------//

void RigidBody:: RigidBody_initialize ()
{
   hmatrix.set_identity ();
   time_of_posture = 0.0;
       x_dot       = 0.0;
       y_dot       = 0.0;
       z_dot       = 0.0;
     phi_dot       = 0.0;
   theta_dot       = 0.0;
     psi_dot       = 0.0;
}
//---------------------------------------------------------------------------//

void RigidBody:: set_velocities (double new_x_dot,
                                 double new_y_dot,
                                 double new_z_dot,
                                 double new_phi_dot,
                                 double new_theta_dot,
                                 double new_psi_dot)
{
       x_dot = new_x_dot;
       y_dot = new_y_dot;
       z_dot = new_z_dot;
     phi_dot = new_phi_dot;
   theta_dot = new_theta_dot;
     psi_dot = new_psi_dot;
}
//---------------------------------------------------------------------------//

void RigidBody:: set_linear_velocities
                                (double new_x_dot,
                                 double new_y_dot,
                                 double new_z_dot)
{
   x_dot = new_x_dot;
   y_dot = new_y_dot;
   z_dot = new_z_dot;
}
//---------------------------------------------------------------------------//

void RigidBody:: set_angular_velocities
                                (double new_phi_dot,
                                 double new_theta_dot,
                                 double new_psi_dot)
{
     phi_dot = new_phi_dot;
   theta_dot = new_theta_dot;
     psi_dot = new_psi_dot;
}
//---------------------------------------------------------------------------//

void RigidBody:: set_time_of_posture (double new_time_of_posture)
{
   time_of_posture = new_time_of_posture;
}
//---------------------------------------------------------------------------//

void RigidBody:: update_Hmatrix     (double delta_t)
{
```

```
   hmatrix.incremental_rotation    (  phi_dot, theta_dot, psi_dot,   delta_t);
   hmatrix.incremental_translation (Vector3D ( x_dot, y_dot, z_dot), delta_t);
}
//-----------------------------------------------------------------------------//

#endif // RIGIDBODY_C
```

I. *Hmatrix.C* Homogeneous Transform Matrix Mathematics

```
////////////////////////////////////////////////////////////////////////////////
/*
 Program:          Hmatrix.C

 Author:           Don Brutzman

 Revised:          12 October 94

 System:           Iris

 Compiler:         ANSI C++

 Compilation:      irix> make dynatest
                   irix> CC Hmatrix.C -lm -c -g +w

                    -c == Produce binaries only, suppressing the link phase.
                    +w == Warn about all questionable constructs.

 Description:      Homogenous transform matrix class specification
                   All angle parameter values are in radians

                   rotations    are in world coordinate system
                   translations are in world coordinate system

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 References:       Fu, K.S., Gonzalez, R.C., and Lee, C.S.G.,
                   _Robotics: Control, Sensing, Vision and Intelligence_,
                   McGraw-Hill Inc., NY, 1987.

                   Cooke, J.C., Zyda, M.J., Pratt, D.R., and McGhee, R.B.,
                   "NPSNET: Flight Simulation Dynamic Modeling using
                   Quaternions," _PRESENCE_, vol. 1 no. 4, Fall 1992,
                   pp. 404-420.

                   Cooke, Joseph C., "NPSNET: Flight Simulation Dynamic
                   Modeling using Quaternions," masters thesis, Naval
                   Postgraduate School, December 1993.

*/
////////////////////////////////////////////////////////////////////////////////

#ifndef HMATRIX_C
#define HMATRIX_C      // prevent errors if multiple #includes present
#include "Quaternion.C"

class Hmatrix
{
private:

// member data fields

   double   htmatrix [4][4];

public:

// constructors and destructor

   Hmatrix                  ();

   Hmatrix                  (const double       x,
                             const double       y,
```

```
                                    const double       z,
                                    const double     phi,
                                    const double   theta,
                                    const double     psi);

    Hmatrix                        (const Vector3D   position_Vector3D,
                                    const double     phi,
                                    const double   theta,
                                    const double     psi);

    Hmatrix                        (const Vector3D   position_Vector3D,
                                    const Vector3D   eulerangles3D);

    Hmatrix                        (const Vector3D   position_Vector3D,
                                    const Quaternion posture);

    Hmatrix                        (const Hmatrix&   input_hmatrix);

   ~Hmatrix                        ()                { /* null body */ }

 // operators

    Hmatrix&   operator =    (const Hmatrix   &h_rhs);       // assignment
    Hmatrix&   operator *    (const Hmatrix   &h_rhs);       // Hmatrix multiply
    Hmatrix&   operator *    (const Vector3D &v_rhs);        // Vector3D multiply

    friend ostream& operator << (ostream& os, Hmatrix& h);

 // inspection methods

    void        print_hmatrix () const;

    double      x_value       () const;
    double      y_value       () const;
    double      z_value       () const;
    double    phi_value       () const;
    double  theta_value       () const;
    double    psi_value       () const;

    Quaternion quaternion_value
                              () const;

    Vector3D   position       () const;
    Vector3D   camera         () const;
    double     scale          () const;

    double     element        (const int row,          // accessor returns lvalue
                               const int column) const; //   for row/column [1..4]

 // modifying methods

    void       rotate         (const double    phi,
                               const double    theta,
                               const double    psi);

    void       rotate         (const Vector3D rotation3D);

    void       rotate_x       (const double    phi);
    void       rotate_y       (const double    theta);
    void       rotate_z       (const double    psi);

    void       incremental_rotation
                              (const double    phi_dot,
                               const double  theta_dot,
                               const double    psi_dot,
                               const double delta_t  );
```

```
    void        translate       (const double   delta_x,
                                  const double   delta_y,
                                  const double   delta_z);

    void        translate       (const Vector3D translation3D);

    void        incremental_translation
                                (const double   x_dot,  const double y_dot,
                                  const double   z_dot,  const double delta_t);

    void        incremental_translation
                                (const Vector3D velocities3D,
                                  const double   delta_t);

    void        set_identity    ();

    void        set_posture     (const double phi,
                                  const double theta,
                                  const double psi);

    void        set_position    (const Vector3D translation3D);

    void        set_camera      (const Vector3D translation3D);
    void        move_camera     (const Vector3D translation3D);

    void        set_all_scales(const double scale_x,
                                  const double scale_y,
                                  const double scale_z,
                                  const double scale_global);
    void        set_scale       (const double scale_global);

};

////////////////////////////////////////////////////////////////////////////////

// rotation matrix conventions:  Cooke et al. Figure 10

// rotation matrix
#define a1      htmatrix[0][0]
#define a2      htmatrix[1][0]
#define a3      htmatrix[2][0]
#define b1      htmatrix[0][1]
#define b2      htmatrix[1][1]
#define b3      htmatrix[2][1]
#define c1      htmatrix[0][2]
#define c2      htmatrix[1][2]
#define c3      htmatrix[2][2]

// position translation vector
#define p_x     htmatrix[0][3]
#define p_y     htmatrix[1][3]
#define p_z     htmatrix[2][3]

// camera perspective transformation
#define c_x     htmatrix[3][0]
#define c_y     htmatrix[3][1]
#define c_z     htmatrix[3][2]

// global scale coefficient
#define hscale htmatrix[3][3]


//--------------------------------------------------------------------------//
// constructor methods
//--------------------------------------------------------------------------//

Hmatrix:: Hmatrix () // default constructor
```

```
{
   set_identity ();
}
//--------------------------------------------------------------------------//

Hmatrix:: Hmatrix (const double      x,
                   const double      y,
                   const double      z,
                   const double    phi,
                   const double  theta,
                   const double    psi) // constructor
{
   set_identity ();
   rotate       (phi, theta, psi);
   translate    (Vector3D (x, y, z));
}
//--------------------------------------------------------------------------//

Hmatrix:: Hmatrix (const Vector3D   position_Vector3D,
                   const double     phi,
                   const double   theta,
                   const double     psi)
{
   set_identity ();
   rotate       (phi, theta, psi);
   translate    (position_Vector3D);
}
//--------------------------------------------------------------------------//

Hmatrix:: Hmatrix (const Vector3D position_Vector3D,
                   const Vector3D eulerangles3D)
{
   set_identity ();
   rotate       (eulerangles3D);
   translate    (position_Vector3D);
}
//--------------------------------------------------------------------------//

Hmatrix:: Hmatrix (const Vector3D position_Vector3D, const Quaternion posture)
{
   set_identity ();
   rotate       (posture.euler_angles ());
   translate    (position_Vector3D);
}
//--------------------------------------------------------------------------//

Hmatrix:: Hmatrix (const Hmatrix& input_hmatrix)
{
   for (int row = 0; row <= 3; row++)
   {
       for (int column = 0; column <= 3; column++)
       {
           htmatrix [row][column] = input_hmatrix.element (row+1, column+1);
       }
   }
}
//--------------------------------------------------------------------------//
// operators
//--------------------------------------------------------------------------//

Hmatrix& Hmatrix:: operator = (const Hmatrix &h_rhs)        // assignment
{
   for (int row = 0; row <= 3; row++)
   {
       for (int column = 0; column <= 3; column++)
       {
           htmatrix [row][column] = h_rhs.htmatrix [row][column];
```

```
        }
    }
    return *this;
}
//-----------------------------------------------------------------------//

Hmatrix& Hmatrix:: operator * (const Hmatrix &h_rhs)    // Hmatrix multiplication
{
    static Hmatrix hresult = Hmatrix ();

    int row, column, index;

    for (row = 0; row <= 3; row++)
    {
        for (column = 0; column <= 3; column++)
        {
            hresult.htmatrix [row][column] = 0.0;
        }
    }

    for (row = 0; row <= 3; row++)
    {
        for (column = 0; column <= 3; column++)
        {
            for (index = 0; index <= 3; index++)
            {
                hresult.htmatrix [row][column] = hresult.htmatrix [row][column] +
                        htmatrix [row][index]  * h_rhs.htmatrix [index][column];
            }
        }
    }
    return hresult;
}
//-----------------------------------------------------------------------//

Hmatrix& Hmatrix:: operator * (const Vector3D &v_rhs)       // Vector3D multiply
{
    static Hmatrix hresult = Hmatrix ();

    int     row, column;
    double position_vector [4];

    position_vector [0] = v_rhs [1];
    position_vector [1] = v_rhs [2];
    position_vector [2] = v_rhs [3];
    position_vector [3] = 1.0;

    for (row = 0; row <= 3; row++)
    {
        for (column = 0; column <= 3; column++)
        {
            hresult.htmatrix [row][column] = 0.0;
        }
    }

    for (row = 0; row <= 3; row++)
    {
        for (column = 0; column <= 3; column++)
        {
            hresult.htmatrix [row][column] =
                    htmatrix [row][column] * position_vector [column];
        }
    }
    return hresult;
}
//-----------------------------------------------------------------------//
```

```
ostream& operator << (ostream& out, Hmatrix& h)
{
   int row;

   for (row = 0; row <= 3; row++)
   {
       out  << endl << "["
            << h.htmatrix [row][0] << ", " << h.htmatrix [row][1] << ", "
            << h.htmatrix [row][2] << ", " << h.htmatrix [row][3] << "]";
   }
   out << endl;
   return (out);
}
//--------------------------------------------------------------------------//
// inspection methods
//--------------------------------------------------------------------------//

void Hmatrix:: print_hmatrix ()
const
{
   int row;

   for (row = 0; row <= 3; row++)
   {
       cout << endl << "["
            << htmatrix [row][0] << ", " << htmatrix [row][1] << ", "
            << htmatrix [row][2] << ", " << htmatrix [row][3] << "]";
   }
   cout << endl;
}
//--------------------------------------------------------------------------//

double Hmatrix:: phi_value ()                                // Cooke (8.17)
const
{
   double theta  = theta_value ();
   double result;

   if ((theta == radians ( 90.0)) || (theta == radians (-90.0)))
   {
      cout << "phi_value () warning: theta == " << theta << endl;
   }

   result = acos (arcclamp (c3 / cos (theta))) * sign (b3);
   return result;
}

//--------------------------------------------------------------------------//

double Hmatrix:: theta_value ()                              // Cooke (8.14)
const
{
   return asin (arcclamp (-a3));
}

//--------------------------------------------------------------------------//

double Hmatrix:: psi_value ()                                // Cooke (8.16)
const
{
   double theta  = theta_value ();
   double result;

   if ((theta == radians ( 90.0)) || (theta == radians (-90.0)))
   {
      cout << "psi_value () warning: theta == " << theta << endl;
   }
```

```
   result = acos (arcclamp (a1 / cos (theta_value ()))) * sign (a2);
   return result;
}

//---------------------------------------------------------------------------//

double Hmatrix:: x_value ()
const
{
   return p_x;
}

//---------------------------------------------------------------------------//

double Hmatrix:: y_value ()
const
{
   return p_y;
}

//---------------------------------------------------------------------------//

double Hmatrix:: z_value ()
const
{
   return p_z;
}

//---------------------------------------------------------------------------//

Quaternion Hmatrix:: quaternion_value ()
const
{
   Quaternion qresult = Quaternion ();
/*
   double     q0, q1, q2, q3;

   q0 = (0.5) * sqrt (a1 + b2 + c3 + 1);
   q1 = 0.0;
   q2 = 0.0;
   q3 = 0.0;

   // <<<<<<<< Shoemake p. 253, FUNDA or p.41 cooke <<<<<<<<<<
*/
   return qresult;
}

//---------------------------------------------------------------------------//

Vector3D Hmatrix:: position ()
const
{
   return Vector3D (p_x, p_y, p_z);
}

//---------------------------------------------------------------------------//

Vector3D Hmatrix:: camera ()
const
{
   return Vector3D (c_x, c_y, c_z);
}

//---------------------------------------------------------------------------//

double Hmatrix:: scale ()
const
```

```
   }
   return hscale;
}

//----------------------------------------------------------------------//

double Hmatrix:: element (const int row,       // accessor returns value
                          const int column)    // with row/column [1..4]
const
{
   if ((row < 1) || (row > 4) || (column < 1) || (column > 4))
   {
      cout << "Warning: Hmatrix.element (" << row << ", " << column <<
              ") is an invalid accessor (only 1..4 allowed), "     <<
              " returning value of 0.0" << endl;

      static double dummy_value = 0.0;
      return        dummy_value;
   }

   return htmatrix [row-1][column-1];
}
//----------------------------------------------------------------------//
// modifying methods
//----------------------------------------------------------------------//

void Hmatrix:: rotate (const double phi, const double theta, const double psi)
{
   double hrotation [3][3];
   double hresult   [3][3];

   int row, column, index;    // reference:  Cooke et al. Figure 10

   double  sinphi   = sin (phi);
   double  cosphi   = cos (phi);
   double  sintheta = sin (theta);
   double  costheta = cos (theta);
   double  sinpsi   = sin (psi);
   double  cospsi   = cos (psi);

   hrotation [0][0] =   costheta * cospsi;                              // a1
   hrotation [1][0] =   costheta * sinpsi;                              // a2
   hrotation [2][0] = - sintheta;                                       // a3
   hrotation [0][1] =   sinphi   * sintheta * cospsi - cosphi * sinpsi; // b1
   hrotation [1][1] =   sinphi   * sintheta * sinpsi + cosphi * cospsi; // b2
   hrotation [2][1] =   sinphi   * costheta;                            // b3
   hrotation [0][2] =   cosphi   * sintheta * cospsi + sinphi * sinpsi; // c1
   hrotation [1][2] =   cosphi   * sintheta * sinpsi - sinphi * cospsi; // c2
   hrotation [2][2] =   cosphi   * costheta;                            // c3

   for (row = 0; row <= 2; row++)
   {
       for (column = 0; column <= 2; column++)
       {
           hresult [row][column] = 0.0; // zero accumulators first

           for (index = 0; index <= 2; index++)
           {
               hresult [row][column] = hresult [row][column] +
                          htmatrix [row][index] * hrotation [index][column];
           }
       }
   }
   for (row = 0; row <= 2; row++)
   {
       for (column = 0; column <= 2; column++)
       {
```

```
                htmatrix [row][column] = hresult [row][column];
            }
        }
}
//---------------------------------------------------------------------------//

void Hmatrix:: rotate (const Vector3D rotation3D)
{
     rotate (rotation3D [1], rotation3D [2],rotation3D [3]);  // note not 0 1 2
}
//---------------------------------------------------------------------------//

void Hmatrix:: rotate_x (const double phi)
{
   double hrotation [3][3];
   double hresult   [3][3];

   int row, column, index;

   for (row = 0; row <= 2; row++)
   {
       for (column = 0; column <= 2; column++)
       {
           if  (row == column) hrotation [row][column] = 1.0;
           else                hrotation [row][column] = 0.0;
       }
   }
   hrotation [1][1] =   cos (phi);
   hrotation [2][2] =   cos (phi);
   hrotation [2][1] =   sin (phi);
   hrotation [1][2] = - sin (phi);

   for (row = 0; row <= 2; row++)
   {
       for (column = 0; column <= 2; column++)
       {
           hresult [row][column] = 0.0; // zero accumulators first

           for (index = 0; index <= 2; index++)
           {
               hresult [row][column] = hresult [row][column] +
                              htmatrix [row][index] * hrotation [index][column];
           }
       }
   }
   for (row = 0; row <= 2; row++)
   {
       for (column = 0; column <= 2; column++)
       {
           htmatrix [row][column] = hresult [row][column];
       }
   }
}
//---------------------------------------------------------------------------//

void Hmatrix:: rotate_y (const double theta)
{
   double hrotation [3][3];
   double hresult   [3][3];

   int row, column, index;

   for (row = 0; row <= 2; row++)
   {
       for (column = 0; column <= 2; column++)
       {
           if  (row == column) hrotation [row][column] = 1.0;
```

```
        else                    hrotation [row][column] = 0.0;
      }
   }
   hrotation [0][0] =   cos (theta);
   hrotation [2][2] =   cos (theta);
   hrotation [2][0] = - sin (theta);
   hrotation [0][2] =   sin (theta);

   for (row = 0; row <= 2; row++)
   {
      for (column = 0; column <= 2; column++)
      {
         hresult [row][column] = 0.0; // zero accumulators first

         for (index = 0; index <= 2; index++)
         {
            hresult [row][column] = hresult [row][column] +
                           htmatrix [row][index] * hrotation [index][column];
         }
      }
   }
   for (row = 0; row <= 2; row++)
   {
      for (column = 0; column <= 2; column++)
      {
         htmatrix [row][column] = hresult [row][column];
      }
   }
}
//---------------------------------------------------------------------------//

void Hmatrix:: rotate_z (const double psi)
{
   double hrotation [3][3];
   double hresult   [3][3];

   int row, column, index;

   for (row = 0; row <= 2; row++)
   {
      for (column = 0; column <= 2; column++)
      {
         if   (row == column) hrotation [row][column] = 1.0;
         else                 hrotation [row][column] = 0.0;
      }
   }
   hrotation [0][0] =   cos (psi);
   hrotation [1][1] =   cos (psi);
   hrotation [1][0] =   sin (psi);
   hrotation [0][1] = - sin (psi);

   for (row = 0; row <= 2; row++)
   {
      for (column = 0; column <= 2; column++)
      {
         hresult [row][column] = 0.0; // zero accumulators first

         for (index = 0; index <= 2; index++)
         {
            hresult [row][column] = hresult [row][column] +
                           htmatrix [row][index] * hrotation [index][column];
         }
      }
   }
   for (row = 0; row <= 2; row++)
   {
      for (column = 0; column <= 2; column++)
```

```
        {
            htmatrix [row][column] = hresult [row][column];
        }
    }
}
//------------------------------------------------------------------------//

void Hmatrix:: incremental_rotation (const double   phi_dot,
                                     const double theta_dot,
                                     const double   psi_dot,
                                     const double delta_t   )
{
     rotate (phi_dot * delta_t, theta_dot * delta_t, psi_dot * delta_t);
}
//------------------------------------------------------------------------//

void Hmatrix:: translate (const double delta_x,
                          const double delta_y,
                          const double delta_z)
{
    p_x += delta_x;
    p_y += delta_y;
    p_z += delta_z;
}
//------------------------------------------------------------------------//

void Hmatrix:: translate (const Vector3D translation3D)
{
    p_x += translation3D [1];
    p_y += translation3D [2];
    p_z += translation3D [3];
}
//------------------------------------------------------------------------//

void Hmatrix:: incremental_translation (const double x_dot,
                                        const double y_dot,
                                        const double z_dot,
                                        const double delta_t)
{
    p_x += x_dot * delta_t;
    p_y += y_dot * delta_t;
    p_z += z_dot * delta_t;
}
//------------------------------------------------------------------------//

void Hmatrix:: incremental_translation (const Vector3D velocities3D,
                                        const double   delta_t)
{
    p_x += velocities3D [1] * delta_t;
    p_y += velocities3D [2] * delta_t;
    p_z += velocities3D [3] * delta_t;
}
//------------------------------------ ------------------------------------//

void Hmatrix:: set_identity () // default constructorset to identity matrix
{
   for (int row = 0; row <= 3; row++)
   {
       for (int column = 0; column <= 3; column++)
       {
         if   (row == column) htmatrix [row][column] = 1.0;
         else                 htmatrix [row][column] = 0.0;
       }
   }
}
//------------------------------------------------------------------------//
```

```
void Hmatrix:: set_posture (const double phi,
                            const double theta,
                            const double psi)
{
     set_identity ();
     rotate (phi, theta, psi);
}
//-------------------------------------------------------------------------//

void Hmatrix:: set_position  (const Vector3D translation3D)
{
     p_x = translation3D [1];
     p_y = translation3D [2];
     p_z = translation3D [3];
}
//-------------------------------------------------------------------------//

void Hmatrix:: set_camera    (const Vector3D translation3D)
{
     c_x = translation3D [1];
     c_y = translation3D [2];
     c_z = translation3D [3];
}
//-------------------------------------------------------------------------//

void Hmatrix:: move_camera    (const Vector3D translation3D)
{
     c_x += translation3D [1];
     c_y += translation3D [2];
     c_z += translation3D [3];
}
//-------------------------------------------------------------------------//

void Hmatrix:: set_all_scales (const double scale_x,
                               const double scale_y,
                               const double scale_z,
                               const double scale_global)
{
     a1     *= scale_x;
     b2     *= scale_y;
     c3     *= scale_z;
     hscale *= scale_global;
}
//-------------------------------------------------------------------------//

void Hmatrix:: set_scale (const double scale_global)
{
     hscale *= scale_global;
}
///////////////////////////////////////////////////////////////////////////
// restore defines

#undef a1
#undef a2
#undef a3
#undef b1
#undef b2
#undef b3
#undef c1
#undef c2
#undef c3
#undef p_x
#undef p_y
#undef p_z
#undef c_x
#undef c_y
#undef c_z
```

```
#undef hscale
/////////////////////////////////////////////////////////////////////////////////////
#endif    // #ifndef HMATRIX_C
```

J. *Quaternion.C* Quaternion Mathematics

```
//////////////////////////////////////////////////////////////////////////////////
/*
 Program:          Quaternion.C

 Revisions:        Don Brutzman

 Revised:          29 SEP 94

 System:           Irix

 Compiler:         ANSI C++

 Compilation:      irix> make dynatest
                   irix> CC Quaternion.C -lm -c -g +w

                    -c == Produce binaries only, suppressing the link phase.
                    +w == Warn about all questionable constructs.

 Description:      Quaternion class specifications and implementation
                   All angle parameter values are in radians.

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 References:       Haynes, Keith, "Computer Graphics Tools for the
                   Visualization of Spacecraft Dynamics," masters thesis,
                   Naval Postgraduate School, December 1993.  Includes
                   source code used for initial version of quaternion.c

                   Cooke, J.C., Zyda, M.J., Pratt, D.R., and McGhee, R.B.,
                   "NPSNET:  Flight Simulation Dynamic Modeling using
                   Quaternions," _PRESENCE_, vol. 1 no. 4, Fall 1992,
                   pp. 404-420.

                   Cooke, Joseph C., "NPSNET:  Flight Simulation Dynamic
                   Modeling using Quaternions," masters thesis, Naval
                   Postgraduate School, December 1993.

                   Chou, Jack C.K., "Quaternion Kinematic and Dynamic
                   Differential Equations," IEEE Transactions on Robotics
                   and Automation, vol. 8 no. 1, February 1992, pp.53-64.

                   Funda, Janez, Taylor, Russell, and Paul, Richard P.,
                   "On Homogenous Transforms, Quaternions, and Computational
                   Efficiency," IEEE Transactions on Robotics and Automation,
                   vol. 6 no. 3, June 1990, pp. 382-388.

                   Shoemake, Ken, "Animating Rotation with Quaternion Curves,"
                   Association for Computing Machinery _SIGGRAPH_ Proceedings,
                   vol. 19 no. 3, July 22-26 1985, pp. 245-254.

*/
//////////////////////////////////////////////////////////////////////////////////

#ifndef  QUATERNION_C
#define  QUATERNION_C     // prevent errors if multiple #includes present

#include "Vector3D.C"

#define  PI     3.1415926535897932

#ifdef TRUE
#undef TRUE
#endif
```

```
#ifdef FALSE
#undef FALSE
#endif

#define  TRUE  1
#define  FALSE 0

/////////////////////////////////////////////////////////////////////////////////

// utility function prototypes

double   sign       (double x);

double   degrees    (double x); // radians input

double   radians    (double x); // degrees input

double   arcclamp   (double x);

double   dnormalize (double angle_radians);  // returns 0..2PI

int      inormalize (double angle_radians)   // returns 0..359
                    {return (int) (degrees (angle_radians) + 0.5) % 360;}

/////////////////////////////////////////////////////////////////////////////////

class Quaternion
{
private:

// member data fields

   double     q0;
   double     q1;
   double     q2;
   double     q3;

public:

// constructors and destructor

   Quaternion                  ();

   Quaternion                  (const double phi,
                                const double theta,
                                const double psi);

   Quaternion                  (const double new_q0,
                                const double new_q1,
                                const double new_q2,
                                const double new_q3);

   Quaternion                  (const Quaternion& q);

  ~Quaternion()                { /* null body */ }

// operators

   Quaternion& operator =      (const  Quaternion& q_rhs);
   Quaternion  operator +      (const  Quaternion& q_rhs);
   Quaternion  operator -      (const  Quaternion& q_rhs);
   Quaternion  operator *      (const  Quaternion& q_rhs);
   Quaternion  operator *      (double scalar);
   double&     operator []     (int);

   friend ostream& operator << (ostream& out, Quaternion& q);
```

```
// inspection methods

   void         print           ();

   double         phi_value     () const; // Euler angle roll
   double       theta_value     () const; // Euler angle pitch
   double         psi_value     () const; // Euler angle yaw

   double       magnitude       () const; // Normally 1.0, unit quaternions
   Vector3D     euler_angles    () const; // Euler angle  phi,  theta, psi
                                          //              (roll, pitch, yaw)
                                          // more efficient than individual calls
   Quaternion   conjugate       () const; // negates vector part
   Quaternion   inverse         () const; // negates vector part & normalizes

// modifying methods

   void         rotate          (const double delta_phi,
                                 const double delta_theta,
                                 const double delta_psi   );

   void         incremental_rotate (const double P, const double Q,
                                    const double R, const double delta_t);

   void         update          (double   P,     double Q,     double R,
                                                               double seconds);

   void         set             (double   phi,  double theta, double psi,
                                                               double rotation);

   void         normalize       ();

};

/////////////////////////////////////////////////////////////////////////////

double sign (double x)
{
   if      (x > 0.0) return  1.0;
   else if (x < 0.0) return -1.0;
   else              return  1.0;
}

//----------------------------------------------------------------------//

double degrees (double x)     // radians input
{
   return x * 180.0 / PI;
}

//----------------------------------------------------------------------//

double radians (double x)    // degrees input
{
   return x * PI / 180.0;
}

//----------------------------------------------------------------------//

double arcclamp (double x)
{
   if      (x >  1.0)
   {
      x =  1.0;
//       cout << " arcclamp reduced " << x << " to 1.0" << endl;
   }
   else if (x < -1.0)
   {
```

```
      x = -1.0;
//      cout << " arcclamp raised " << x << " to -1.0" << endl;
   }
   return x;
}
//------------------------------------------------------------------------//

double dnormalize (double angle_radians)
{
   double new_angle = angle_radians;

   while (new_angle > 2*PI) new_angle -= 2*PI;
   while (new_angle < 0.0 ) new_angle += 2*PI;
   return new_angle;
}
//------------------------------------------------------------------------//
// constructor methods
//------------------------------------------------------------------------//

Quaternion:: Quaternion ()
{
   q0 = 1.0;
   q1 = 0.0;
   q2 = 0.0;
   q3 = 0.0;
}
//------------------------------------------------------------------------//

Quaternion:: Quaternion (const double phi,
                         const double theta,
                         const double psi)
{
   // reference:  Cooke thesis p. 41

   double r, p, y, cosr, cosp, cosy, sinr, sinp, siny;

   r    = (phi / 2.0);    p    = (theta / 2.0);    y    = (psi / 2.0);
   cosr = cos (r);        cosp = cos (p);          cosy = cos (y);
   sinr = sin (r);        sinp = sin (p);          siny = sin (y);
   q0 =   (cosr * cosp * cosy) + (sinr * sinp * siny);
   q1 =   (sinr * cosp * cosy) - (cosr * sinp * siny);
   q2 =   (cosr * sinp * cosy) + (sinr * cosp * siny);
   q3 = - (sinr * sinp * cosy) + (cosr * cosp * siny);
}
//------------------------------------------------------------------------//

Quaternion:: Quaternion (const double new_q0,
                         const double new_q1,
                         const double new_q2,
                         const double new_q3)
{
   q0 = new_q0;
   q1 = new_q1;
   q2 = new_q2;
   q3 = new_q3;
}
//------------------------------------------------------------------------//

Quaternion:: Quaternion (const Quaternion& q)
{
   q0 = q.q0;
   q1 = q.q1;
   q2 = q.q2;
   q3 = q.q3;
}
//------------------------------------------------------------------------//
// operators
```

```
//--------------------------------------------------------------------------//

Quaternion& Quaternion:: operator = (const Quaternion& q_rhs)
{
   q0 = q_rhs.q0;
   q1 = q_rhs.q1;
   q2 = q_rhs.q2;
   q3 = q_rhs.q3;
   return *this;
}
//--------------------------------------------------------------------------//

Quaternion Quaternion:: operator + (const Quaternion& q_rhs)
{
   static Quaternion sum;

   sum.q0 = q0 + q_rhs.q0;
   sum.q1 = q1 + q_rhs.q1;
   sum.q2 = q2 + q_rhs.q2;
   sum.q3 = q3 + q_rhs.q3;
   return sum;
}
//--------------------------------------------------------------------------//

Quaternion Quaternion:: operator - (const Quaternion& q_rhs)
{
   static Quaternion difference;

   difference.q0 = q0 - q_rhs.q0;
   difference.q1 = q1 - q_rhs.q1;
   difference.q2 = q2 - q_rhs.q2;
   difference.q3 = q3 - q_rhs.q3;
   return difference;
}
//--------------------------------------------------------------------------//

Quaternion Quaternion:: operator * (const Quaternion& q_rhs)
{
   static Quaternion prod;
   prod.q0 = (q0 * q_rhs.q0) - (q1 * q_rhs.q1) -
             (q2 * q_rhs.q2) - (q3 * q_rhs.q3);
   prod.q1 = (q1 * q_rhs.q0) + (q0 * q_rhs.q1) -
             (q3 * q_rhs.q2) + (q2 * q_rhs.q3);
   prod.q2 = (q2 * q_rhs.q0) + (q3 * q_rhs.q1) -
             (q0 * q_rhs.q2) + (q1 * q_rhs.q3);
   prod.q3 = (q3 * q_rhs.q0) + (q2 * q_rhs.q1) -
             (q1 * q_rhs.q2) + (q0 * q_rhs.q3);
   return prod;
}
//--------------------------------------------------------------------------//

Quaternion Quaternion:: operator * (double scalar)
{
   static Quaternion product;

   product.q0 = q0 * scalar;
   product.q1 = q1 * scalar;
   product.q2 = q2 * scalar;
   product.q3 = q3 * scalar;
   return product;
}
//--------------------------------------------------------------------------//

double& Quaternion:: operator [] (int n)
{
   if (n == 0)
   {
```

```
      return q0;
   }
   if (n == 1)
   {
      return q1;
   }
   if (n == 2)
   {
      return q2;
   }
   if (n == 3)
   {
      return q3;
   }
   cout << "Warning: quaternion[" << n << "] is an invalid accessor"
        << " (only 0..3 allowed), returning value of 0.0" << endl;
   static double dummy_value = 0.0;
   return dummy_value;
}
//--------------------------------------------------------------------------//

ostream& operator << (ostream& out, Quaternion& q)
{
   out << "[" << q.q0 << ", " << q.q1 << ", "
              << q.q2 << ", " << q.q3 << "]" ;
   return (out);
}
//--------------------------------------------------------------------------//
// inspection methods
//--------------------------------------------------------------------------//

void Quaternion:: print ()
{
   cout << "[" << q0 << ", " << q1 << ", "
               << q2 << ", " << q3 << "]" ;
}
//--------------------------------------------------------------------------//

double Quaternion:: phi_value ()    // Euler angle roll

const
{
   return   acos (arcclamp ((q0*q0 - q1*q1 - q2*q2 + q3*q3)
                            / cos (theta_value ())))

          * sign  (q2*q3 + q0*q1);
}
//--------------------------------------------------------------------------//

double Quaternion:: theta_value ()  // Euler angle pitch

const
{
   return asin (arcclamp (-2.0 * (q1*q3 - q0*q2)));
}
//--------------------------------------------------------------------------//

double Quaternion:: psi_value ()    // Euler angle yaw

const
{
   return   acos (arcclamp ((q0*q0 + q1*q1 - q2*q2 - q3*q3)
                            / cos (theta_value ())))

          * sign  (q1*q2 + q0*q3);
}
//--------------------------------------------------------------------------//
```

```
double Quaternion:: magnitude ()

const
{
   return sqrt((q0 * q0) + (q1 * q1) + (q2 * q2) + (q3 * q3));
}
//--------------------------------------------------------------------------//

Vector3D Quaternion:: euler_angles ()     // Euler angle  phi,  theta, psi
                                          //              (roll, pitch, yaw)
                                          // more efficient than individual calls
const
{
   double qq0, qq1, qq2, qq3, phi, theta, psi, costheta; // locals hide methods

   qq0      = q0 * q0; // force optimization of squaring computations
   qq1      = q1 * q1;
   qq2      = q2 * q2;
   qq3      = q3 * q3;

   theta    = asin (arcclamp (-2.0 * (q1*q3 - q0*q2)));

   costheta = cos (theta);

   phi      =    acos (arcclamp ((qq0 - qq1 - qq2 + qq3) / costheta))

               * sign  (q2*q3 + q0*q1);

   psi      =    acos (arcclamp ((qq0 + qq1 - qq2 - qq3) / costheta))

               * sign  (q1*q2 + q0*q3);

   return Vector3D (phi, theta, psi);
}
//--------------------------------------------------------------------------//

Quaternion Quaternion:: conjugate ()    // negates vector part

const
{
   return Quaternion (q0, - q1, - q2, - q3);
}
//--------------------------------------------------------------------------//

Quaternion Quaternion:: inverse ()    // negates vector part

const
{
   static Quaternion qresult;

   qresult = conjugate ();
   qresult.normalize ();
   return qresult;
}
//--------------------------------------------------------------------------//
// modifying methods
//--------------------------------------------------------------------------//

void Quaternion:: rotate (const double delta_phi,
                          const double delta_theta,
                          const double delta_psi   )
{
   Quaternion rotation;

   rotation.q0= -0.5*((q1 * delta_phi)  +(q2 * delta_theta)+(q3 * delta_psi));
   rotation.q1=  0.5*((q0 * delta_phi)  +(q2 * delta_psi)  -(q3 * delta_theta));
   rotation.q2=  0.5*((q0 * delta_theta)+(q3 * delta_phi)  -(q1 * delta_psi));
```

```
   rotation.q3=  0.5*((q0 * delta_psi)  +(q1 * delta_theta)-(q2 * delta_phi));

   q0 += rotation.q0;
   q1 += rotation.q1;
   q2 += rotation.q2;
   q3 += rotation.q3;
   normalize ();
   return;
}
//---------------------------------------------------------------------------//

void Quaternion:: incremental_rotate (const double P, const double Q,
                                      const double R, const double delta_t)
{
   Quaternion rotation;

   rotation.q0 = -0.5 * delta_t * ((q1 * P) + (q2 * Q) + (q3 * R));
   rotation.q1 =  0.5 * delta_t * ((q0 * P) + (q2 * R) - (q3 * Q));
   rotation.q2 =  0.5 * delta_t * ((q0 * Q) + (q3 * P) - (q1 * R));
   rotation.q3 =  0.5 * delta_t * ((q0 * R) + (q1 * Q) - (q2 * P));

   q0 += rotation.q0;
   q1 += rotation.q1;
   q2 += rotation.q2;
   q3 += rotation.q3;
   normalize ();
   return;
}
//---------------------------------------------------------------------------//

void Quaternion:: update (double P, double Q, double R, double seconds)
{
   double      hh = seconds * 0.5;
   double      h6 = seconds / 6.0;
   Quaternion y  = *this, dym, dyt, yt, dydx;

   dydx.q0 = -0.5 * ((q1 * P) + (q2 * Q) + (q3 * R));
   dydx.q1 =  0.5 * ((q0 * P) + (q2 * R) - (q3 * Q));
   dydx.q2 =  0.5 * ((q0 * Q) + (q3 * P) - (q1 * R));
   dydx.q3 =  0.5 * ((q0 * R) + (q1 * Q) - (q2 * P));

   yt.q0 = y.q0 + hh * dydx.q0;
   yt.q1 = y.q1 + hh * dydx.q1;
   yt.q2 = y.q2 + hh * dydx.q2;
   yt.q3 = y.q3 + hh * dydx.q3;

   dyt.q0 = -0.5 * ((yt.q1 * P) + (yt.q2 * Q) + (yt.q3 * R));
   dyt.q1 =  0.5 * ((yt.q0 * P) + (yt.q2 * R) - (yt.q3 * Q));
   dyt.q2 =  0.5 * ((yt.q0 * Q) + (yt.q3 * P) - (yt.q1 * R));
   dyt.q3 =  0.5 * ((yt.q0 * R) + (yt.q1 * Q) - (yt.q2 * P));

   yt.q0 = y.q0 + hh * dyt.q0;
   yt.q1 = y.q1 + hh * dyt.q1;
   yt.q2 = y.q2 + hh * dyt.q2;
   yt.q3 = y.q3 + hh * dyt.q3;

   dym.q0 = -0.5 * ((yt.q1 * P) + (yt.q2 * Q) + (yt.q3 * R));
   dym.q1 =  0.5 * ((yt.q0 * P) + (yt.q2 * R) - (yt.q3 * Q));
   dym.q2 =  0.5 * ((yt.q0 * Q) + (yt.q3 * P) - (yt.q1 * R));
   dym.q3 =  0.5 * ((yt.q0 * R) + (yt.q1 * Q) - (yt.q2 * P));

   yt.q0 = y.q0 + seconds * dym.q0;
   yt.q1 = y.q1 + seconds * dym.q1;
   yt.q2 = y.q2 + seconds * dym.q2;
   yt.q3 = y.q3 + seconds * dym.q3;

   dym.q0 = dym.q0 + dyt.q0;
```

```
    dym.q1 = dym.q1 + dyt.q1;
    dym.q2 = dym.q2 + dyt.q2;
    dym.q3 = dym.q3 + dyt.q3;

    dyt.q0 = -0.5 * ((yt.q1 * P) + (yt.q2 * Q) + (yt.q3 * R));
    dyt.q1 =  0.5 * ((yt.q0 * P) + (yt.q2 * R) - (yt.q3 * Q));
    dyt.q2 =  0.5 * ((yt.q0 * Q) + (yt.q3 * P) - (yt.q1 * R));
    dyt.q3 =  0.5 * ((yt.q0 * R) + (yt.q1 * Q) - (yt.q2 * P));

    q0 = y.q0 + h6 * (dydx.q0 + dyt.q0 + 2.0 * dym.q0);
    q1 = y.q1 + h6 * (dydx.q1 + dyt.q1 + 2.0 * dym.q1);
    q2 = y.q2 + h6 * (dydx.q2 + dyt.q2 + 2.0 * dym.q2);
    q3 = y.q3 + h6 * (dydx.q3 + dyt.q3 + 2.0 * dym.q3);

    normalize ();
}
//------------------------------------------------------------------------//

void Quaternion:: set (double phi, double theta, double psi, double twist)
{
    q0 =                cos (0.5 * twist);
    q1 = cos (phi)    * sin (0.5 * twist);
    q2 = cos (theta)  * sin (0.5 * twist);
    q3 = cos (psi)    * sin (0.5 * twist);
}
//------------------------------------------------------------------------//

void Quaternion:: normalize()
{
double m = magnitude ();
if (m > 0.0)
{
     q0 /= m;
     q1 /= m;
     q2 /= m;
     q3 /= m;
}
}
//------------------------------------------------------------------------//

////////////////////////////////////////////////////////////////////////////

#endif   // #ifndef QUATERNION_C
```

K. *Vector3D.C* 3D Vector Mathematics

```
//////////////////////////////////////////////////////////////////////////////////////
/*
 Program:          Vector3D.C

 Original Author: Keith Haynes

 Revisions:        Don Brutzman

 Revised:          12 FEB 94

 System:           Iris/PC

 Compiler:         ANSI C++

 Compilation:      irix> CC Vector3D.C -lm -c -g +w

                    -c == Produce binaries only, suppressing the link phase.
                    +w == Warn about all questionable constructs.

 Description:      Vector3D class specifications and implementation

 Advisors:         Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

*/
//////////////////////////////////////////////////////////////////////////////////////

#ifndef VECTOR3D_C
#define VECTOR3D_C      // prevent errors in multiple #includes present

#include <iostream.h>
#include <iomanip.h>
#include <math.h>

class Vector3D
{
// member data fields

        double x;
        double y;
        double z;

public:

// member constructor and destructor functions

        Vector3D               ();
        Vector3D               (double a, double b, double c);
        Vector3D               (const Vector3D&);
       ~Vector3D ()            { /* null body */ }

// operators

        Vector3D & operator =  (const Vector3D&);
        Vector3D   operator +  (const Vector3D&);
        Vector3D   operator -  (const Vector3D&);
        double           operator *   (const Vector3D&);  // dot product
        Vector3D   operator *  (double);             // scalar multiplication
        Vector3D   operator /  (double);             // scalar division
        Vector3D   operator ^  (const Vector3D&);    // cross product
        double &   operator [] (int) const;


// inspection methods
```

```
        double      magnitude   ();
        friend      ostream & operator << (ostream& os, Vector3D& v);
        void        print ();

// modifying methods

        void        normalize   ();
        void        normalize   (double);
};

////////////////////////////////////////////////////////////////////////////////

//default constructor
Vector3D::Vector3D()
{
        x = 0.0;
        y = 0.0;
        z = 0.0;
}

//constructor using three doubles
Vector3D::Vector3D(double a, double b, double c)
{
        x = a;
        y = b;
        z = c;
}

//constructor using another Vector3D
Vector3D::Vector3D(const Vector3D& v)
{
        x = v.x;
        y = v.y;
        z = v.z;
}

//Assignment operator - the function must return a reference to a Vector
//instead of a Vector for assignment to work properly
Vector3D& Vector3D::operator=(const Vector3D& v)
{
        x = v.x;
        y = v.y;
        z = v.z;
        return *this;
}

//Vector addition operator
Vector3D Vector3D::operator+(const Vector3D& v)
{
        Vector3D sum;
        sum.x = v.x + x;
        sum.y = v.y + y;
        sum.z = v.z + z;
        return sum;
}

//Vector substraction
Vector3D Vector3D::operator-(const Vector3D& v)
{
        Vector3D diff;
        diff.x = x - v.x;
        diff.y = y - v.y;
        diff.z = z - v.z;
        return diff;
}

//Vector dot product
```

```
double Vector3D::operator*(const Vector3D& v)
{
        double dot;
        dot = (v.x * x) + (v.y * y) + (v.z * z);
        return dot;
}

//scalar multiplication
Vector3D Vector3D::operator*(double n)
{
        Vector3D mult;
        mult.x = x * n;
        mult.y = y * n;
        mult.z = z * n;
        return mult;
}

//scalar division - it is the user responsibility to make sure that n is not
zero
Vector3D Vector3D::operator/(double n)
{
        Vector3D result;
        result.x = x / n;
        result.y = y / n;
        result.z = z / n;
        return result;
}

//Vector cross product
Vector3D Vector3D::operator^(const Vector3D& v)
{
        Vector3D cross;
        cross.x =   (y * v.z) - (v.y * z);
        cross.y = -((x * v.z) - (v.x * z));
        cross.z =   (x * v.y) - (v.x * y);
        return cross;
}

//the << operator is to be used with output stream out
ostream& operator << (ostream& out, Vector3D& v)
{
   out << "[" << v.x << ", " << v.y << ", " << v.z << "]";
        return (out);
}

void Vector3D:: print ()
{
   cout << "[" << x << ", " << y << ", " << z << "]";
}

//allows access to the components of the Vector3D.  it must return a reference
//in order for assignment to work
double& Vector3D::operator[](int n)
const
{
   static double result;

        if (n == 1)
        {
      result = x;
                return result;
        }
        if (n == 2)
        {
      result = y;
                return result;
        }
```

```
        if (n == 3)
        {
      result = z;
                return result;
        }
        cout << "Warning: Vector3D[" << n << "] is an invalid accessor"
             << " (only 1..3 allowed), returning value of 0.0" << endl;

        static double dummy_value = 0.0;
        return dummy_value;
}

//returns the magnitude of the Vector
double Vector3D::magnitude()
{
        return sqrt((x * x) + (y * y) + (z * z));
}

//normalizes the Vector to one
void Vector3D::normalize()
{
        double m = sqrt((x * x) + (y * y) + (z * z));
        if (m)
        {
        x = x / m;
        y = y / m;
        z = z / m;
        }
}

//normalize the Vector to d
void Vector3D::normalize(double d)
{
        double m = sqrt((x * x) + (y * y) + (z * z));
        if (m)
        {
        x = d * x / m;
        y = d * y / m;
        z = d * z / m;
        }
}

////////////////////////////////////////////////////////////////////////////////

#endif   // #ifndef VECTOR3D_C
```

L. *Makefile* for Hydrodynamics Classes

```
# Makefile dynamics model for underwater virtual world
# 14 October 94    Don Brutzman

SGIIRIX4FLAGS    = -O2

##############################################################################
# DIS includes

INCS             = -I/n/dude/work/brutzman/DIS.mcast/h
LIB_DIR                  = -L/n/dude/work/brutzman/DIS.mcast/src
LIB              = /n/dude/work/brutzman/DIS.mcast/src/libdis_client.a
LIB_ARG                  = -ldis_client   -lmpc -lm
HDR_PATH         = /n/dude/work/brutzman/DIS.mcast/h/
HDRS             = $(HDR_PATH)pdu.h $(HDR_PATH)disdefs.h

##############################################################################

CCFLAGS                  = +w $(SGIIRIX4FLAGS)

#INCS            = -I/n/elsie/work3/macedoni/net/mcast/network/h
#LIB_DIR                 = -L/n/elsie/work3/macedoni/net/mcast/network/bin
#LIB             = /n/elsie/work3/macedoni/net/mcast/network/bin/libdis_client.a
#LIB_ARG                 = -ldis_client -lmpc -lm

#HDR_PATH        = /n/elsie/work3/macedoni/net/mcast/network/h/
#HDRS            = $(HDR_PATH)pdu.h $(HDR_PATH)disdefs.h

#NET_DIR          = /n/elsie/work3/macedoni/net/mcast/network/src/
#NET_DIR          = /n/dude/work/brutzman/DIS.mcast/src
#NETOBJS          = $(NET_DIR)attach.o $(NET_DIR)client_lib.o            \
#                   $(NET_DIR)free.o   $(NET_DIR)mallocs.o $(NET_DIR)sends.o   \
#                   $(NET_DIR)recvs.o  $(NET_DIR)protocol.o

OBJS =  dynamics.o      AUVsocket.o     Vector3D.o          Quaternion.o \
        Hmatrix.o       RigidBody.o     DISNetworkedRigidBody.oUUVBody.o    \
        SonarModel.o    math_utilities.o

##############################################################################

dynamics: $(LIB) $(HDRS) $(OBJS)
        @echo "Linking..."
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(OBJS) $(NETOBJS) $(INCS) $(LIB_ARG) \
           -o dynamics

# Makefile requirement is that no libraries are linked to the objects


Vector3D.o: Vector3D.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c Vector3D.C -o Vector3D.o

Quaternion.o: Quaternion.C Vector3D.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c Quaternion.C -o Quaternion.o

Hmatrix.o: Hmatrix.C Vector3D.C Quaternion.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c Hmatrix.C -o Hmatrix.o
```

```
RigidBody.o: RigidBody.C Hmatrix.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c RigidBody.C -o RigidBody.o

AUVsocket.o: AUVsocket.C AUVglobals.H UUVmodel.H Quaternion.C $(HDRS)
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c AUVsocket.C -o AUVsocket.o

DISNetworkedRigidBody.o: DISNetworkedRigidBody.C AUVglobals.H UUVmodel.H\
            RigidBody.C $(HDRS)
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c DISNetworkedRigidBody.C -o DISNetworkedRigidBody.o

SonarModel.o: SonarModel.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c SonarModel.C -o SonarModel.o

UUVBody.o: UUVBody.C AUVglobals.H UUVmodel.H SonarModel.C AUVsocket.C\
           DISNetworkedRigidBody.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c UUVBody.C -o UUVBody.o

dynamics.o: dynamics.C AUVglobals.H UUVmodel.H UUVBody.C $(HDRS)
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c dynamics.C -o dynamics.o


math_utilities.o: math_utilities.C
        CC $(SGIIRIX4FLAGS) $(LIB_DIR) $(NETOBJS) $(INCS) $(LIB_ARG)\
           -c math_utilities.C -o math_utilities.o

# $(OBJS): AUVglobals.H UUVmodel.H $(HDRS)
#       CC -c $(CCFLAGS) $(INCS) $*.C

# utilities:

delete:
        rm -f core *.o *.a a.out dynamics

clean:
        rm -f core *.o *.a a.out

all: delete dynamics
```

VI. SONAR MODEL

A. *SonarModel.C* Geometric Sonar Model

```
////////////////////////////////////////////////////////////////////////////////
/*
 Program:           SonarModel.C

 Description:       Underwater vehicle geometric sonar model

 Revised:           21 October 94

 System:            Irix

 Compiler:          ANSI C++

 Compilation:       irix> make dynamics
                    irix> CC SonarModel.C -lm -c -g +w

                    -c == Produce binaries only, suppressing the link phase.
                    +w == Warn about all questionable constructs.

 Advisors:          Dr. Mike Zyda, Dr. Bob McGhee and Dr. Tony Healey

 Author:            Don Brutzman                    brutzman@cs.nps.navy.mil
                    Code OR/Br
                    Naval Postgraduate School       (408) 656-2149 work
                    Monterey CA 93943-5000          (408) 656-2595 fax

 References:


 Status:            Only pool geometric model included


 Future work:       Comments and suggestions are welcome!

*/
////////////////////////////////////////////////////////////////////////////////

#ifndef SONARMODEL_C
#define SONARMODEL_C     // prevent errors if multiple #includes present

#include "AUVsocket.C"

int     precede          (double angle1, double angle2);

int     precede_radians (double angle1, double angle2);

void    test_tank_sonar_model ();  // parallelize & generalize this sonar model
                                   // for all objects in vw, using Ziomek's RRA
double radian_normalize  (double rads);
double radian_normalize2 (double rads);

////////////////////////////////////////////////////////////////////////////////

/*---------------------------------------------------------------------------*/
/* excerpt from geometric reasoning circle world  circle.c                   */
```

```
/* Boolean function for angle precedence */

int    precede (double angle1, double angle2)

                                        /* angles are any real angles in degrees */
                                        /* return TRUE if angle1 precedes angle2,*/
                                        /*   return FALSE otherwise              */
{
/* note that the input angles are individually normalized to ensure validity  */

    if (normalize2 (normalize2 (angle2) - normalize2 (angle1)) > 0.0)
       return TRUE;
    else                                /* reference:  equation (7) class notes  */
       return FALSE;
}
/*-----------------------------------------------------------------------------*/

double radian_normalize (double rads)     /* radians input*/
{
   double result = rads;

   while (result <       0.0) result += 2.0 * PI;
   while (result >= 2.0 * PI) result -= 2.0 * PI;

   return result;
}
/*-----------------------------------------------------------------------------*/

double radian_normalize2 (double rads)     /* radians input*/
{
   double result = rads;

   while (result <= - PI) result += 2.0 * PI;
   while (result >    PI) result -= 2.0 * PI;

   return result;
}

/*-----------------------------------------------------------------------------*/
/* excerpt from geometric reasoning circle world  circle.c                     */

/* Boolean function for angle precedence */

int    precede_radians (double angle1, double angle2)

                                        /* angles are any real angles in degrees */
                                        /* return TRUE if angle1 precedes angle2,*/
                                        /*   return FALSE otherwise              */
{
/* note that the input angles are individually normalized to ensure validity  */

    if (radian_normalize2 (radian_normalize2 (angle2) -
                           radian_normalize2 (angle1)) > 0.0)
       return TRUE;
    else                                /* reference:  equation (7) class notes  */
       return FALSE;
}
/*-----------------------------------------------------------------------------*/
```

```
///////////////////////////////////////////////////////////////////////////////

void test_tank_sonar_model ()   // parallelize & generalize this sonar model
{
   double SA, SB, SC, SD;                  // angles from sonar to corners

   // Test tank coordinates:     A (-10,  10)  +--+  (10,  10) B      X-axis
   //                                          |  |                    ^
   //                                          |  |                    |
   //                            D (-10, -10)  +--+  (10, -10) C      +->Y-axis

   double return_x, return_y, distance;  // intersect coordinates & distance

   // AUV_ST1000_bearing and AUV_ST725_bearing determined by vehicle control

                                     // ST_1000 conical pencil beam bearing
   AUV_ST1000_range    = 0.0;        // ST_1000 conical pencil beam range
   AUV_ST1000_strength= 0.0;         // ST_1000 conical pencil beam strength

                                     // ST_725  1 x 24  sector beam bearing
   AUV_ST725_range     = 0.0;        // ST_725  1 x 24  sector beam range
   AUV_ST725_strength = 0.0;         // ST_725  1 x 24  sector beam strength

   double sonar_x    = AUV_x + cos (AUV_psi) * AUV_ST1000_x_offset;
   double sonar_y    = AUV_y + sin (AUV_psi) * AUV_ST1000_x_offset;

// remove these lines when auv vw viewer includes sonar offset
   sonar_x           = AUV_x;
   sonar_y           = AUV_y;

   double sonar_psi = AUV_psi + radians (AUV_ST1000_bearing);

   if ((AUV_x <= 10.0) && (AUV_x >= -10.0) &&
       (AUV_y <= 10.0) && (AUV_y >= -10.0) &&
       (AUV_z <= 50.0) && (AUV_z >=  41.0))
   {
      AUV_ST1000_strength= 10.0;       // return expected
      AUV_ST725_strength = 10.0;       // return expected

      // note that a reversed x,y calling sequence is necessary
      //   in order to get correct quadrant alignment
      SA = atan2 ((-10.0) - sonar_y, ( 10.0) - sonar_x);
      SB = atan2 (( 10.0) - sonar_y, ( 10.0) - sonar_x);
      SC = atan2 (( 10.0) - sonar_y, (-10.0) - sonar_x);
      SD = atan2 ((-10.0) - sonar_y, (-10.0) - sonar_x);

      if (TRACE)
      {
          cout <<   "AUV_x = "    << AUV_x
               << ", AUV_y = "    << AUV_y
               << ", AUV_psi = " << degrees (AUV_psi) << endl;
          cout <<   "sonar_x = "   << sonar_x
               << ", sonar_y = "   << sonar_y
               << ", sonar_psi = " << degrees (sonar_psi) << endl;
          cout <<   "SA = "       << degrees (SA)
               << ", SB = "       << degrees (SB)
               << ", SC = "       << degrees (SC)
               << ", SD = "       << degrees (SD)
               << endl;
          cout <<     "precede_radians (SA, AUV_psi) = "
               <<      precede_radians (SA, AUV_psi) << endl;
      }

      if      (   precede_radians (SA, AUV_psi)
               && precede_radians (AUV_psi, SB))
```

```
    {
        if (TRACE)  cout << "SECTOR I" << endl;
        return_x = 10.0;
        return_y = sonar_y + sin (sonar_psi) * (10.0 - sonar_x);
    }
    else if (   precede_radians (SB, sonar_psi)
            && precede_radians (sonar_psi, SC))
    {
        if (TRACE)  cout << "SECTOR II" << endl;
        return_x = sonar_x - sin (sonar_psi - PI/2.0) * (10.0 - sonar_y);
        return_y = 10.0;
    }
    else if (   precede_radians (SC, sonar_psi)
            && precede_radians (sonar_psi, SD))
    {
        if (TRACE)  cout << "SECTOR III" << endl;
        return_x = - 10.0;
        return_y = sonar_y - sin (sonar_psi - PI) * (sonar_x + 10.0);
    }
    else if (   precede_radians (SD, sonar_psi)
            && precede_radians (sonar_psi, SA))
    {
        if (TRACE)  cout << "SECTOR IV" << endl;
        return_x = sonar_x + sin (sonar_psi + PI/2.0) * (sonar_y + 10.0);
        return_y = - 10.0;
    }
    else cout << "Computational sonar error, no sector was determined."
            << endl;

    distance = sqrt (   (return_x - sonar_x) * (return_x - sonar_x)
                      + (return_y - sonar_y) * (return_y - sonar_y));

    AUV_ST1000_range = distance;
    AUV_ST725_range  = distance;

    if (TRACE)
    {
      cout <<    "return_x = "         << return_x
           << ", return_y = "          << return_y << endl;

      cout <<    "AUV_ST1000_range = " << AUV_ST1000_range
           << ", AUV_ST725_range = "   << AUV_ST725_range << endl;
    }
  }
  else return; // outside of test tank, default return is 0

} // end test_tank_sonar_model

//////////////////////////////////////////////////////////////////////////////////////

#endif // SONARMODEL_C
```

VII. NETWORKING

A. Introduction

The principal networking code used in the underwater virtual world has already been presented in previous listings. Source programs *execution.c* and *AUVsocket.c* implement socket communications between robot and virtual world, while *DISNetworkedRigidBody.C* and *viewer.C* communicate via DIS PDUs over the MBone. These communications links were not created from scratch. The following test programs were used in program development and are included as valid standalone applications that can be used to test network software, independently of the robot or computer graphics applications.

os9sender.c and *os9server.c* are a client/server pair used to open test and close a point-to-point socket. What is unusual about these programs is that they were developed to operate interchangeably on either Unix systems using Berkeley Software Distribution (BSD) 4.3 network routines, or on OS-9 processors using Microware-provided network routines (Microware 91a, 91b). Network code is difficult to debug so extensive trace options are provided to assist in diagnosing difficulties.

Prior to the current multicast version (2.0.3) of the DIS library distribution, all DIS interfaces used broadcast or unicast transport protocols. This meant that all network interactions between DIS applications were restricted to a single local area network (LAN). Such a limitation is unacceptable if an underwater virtual world is to be capable of communicating globally across the Internet. *disbridge.c* was written to create a socket bridge between LANs which reflected DIS PDUs identically in either direction. Multiple *disbridge* programs could connect arbitrary numbers of LANs with only a single *disbridge* needed for each new network addition. Although such an approach is inconvenient and a somewhat inefficient use of bandwidth, it can extend the broadcast/unicast DIS approach to large numbers of networks.

disbridge has been tested successfully with earlier versions of the underwater

virtual world, and also with the NPS Platform Foundation (Bailey 94). Fortunately *disbridge* is no longer needed since the newer multicast DIS version 2.0.3 can take advantage of the MBone to attain Internet-wide connectivity (Zeswitz 93). Nevertheless only about a thousand subnets are currently connected to the MBone, and conceivably *disbridge* might someday be useful to connect networks independently of the MBone. Recommended future work includes upgrading *disbridge* to be compatible with future versions of the DIS library in unicast, broadcast or multicast modes.

B. *os9sender.c* Unix to OS-9 Socket Communications Client

```
/*****************************************************************************
 Program:            os9sender.c

 Description:        socket test program to pass input strings in 2 directions,
                        especially socket from OS-9 auvsim1 to Irix fletch.cs

 Revised:            31 MAY 94

 Compilation:        unix> cc os9sender.c -o os9sender
                     os-9> make os9sender

 Execution:          host1>   os9server
                     host2>   os9sender -r host1

 Example:            fletch>  os9server
                     auvsim1> os9sender -r fletch


 Execution options:

           -r -h -t  (first letters are sufficient, capital letters are OK too)

  -remote hostname   hostname.remote.net.address for client to connect to server
                     no default is present

             -help   display calling syntax

            -trace   turn on TRACE mode


 References:      (1) (Gespac-provided) EVIRA EVLAN-11 Ethernet Data Link
                      Controller for the G64/G96 Bus/EVTCP Internet Package
                      technical manuals
                  (2) Internetworking with TCP/IP Volume I:  Principles,
                      Protocols and Architectures, Douglas E. Comer,
                      Prentice Hall, Englewood Cliffs NJ, 1991
                  (3) Internetworking with TCP/IP Volume II:  Design,
                      Implementation and Internals, Douglas E. Comer and
                      David L. Stevens, Prentice Hall, Englewood Cliffs NJ, 1991
                  (4) IRIX Network Programming Guide, Silicon Graphics Inc.
                  (5) An Advanced 4.3BSD Interprocess Communication Tutorial,
                      Samuel J. Leffler, Robert S. Fabry, William N. Joy,
                      Phil Lapsley, Steve Miller and Chris Torek, undated
                  (6) Real-Time Programming Tutorial, Bill Mannel, SGI Expo,
                      Silicon Graphics Inc., 23 May 93
```

```
Author:             Don Brutzman                      brutzman@cs.nps.navy.mil
                    Code OR/Br
                    Naval Postgraduate School         (408) 656-2149 work
                    Monterey CA 93943-5000            (408) 656-2595 fax

Status:             Tested satisfactory Irix<==>Irix, OS-9<==>OS-9, Irix<==>OS-9


Future work:        Consider implementation as an inetd 'superserver' daemon.

                    Ensure compatibility with DIS v2.0 protocols when available.


                    add #elif (sun) defines

                    Comments, suggestions and corrections are welcome!


*******************************************************************************/
/* Irix defines in /usr/include */
#if defined(sgi)

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <string.h>

/* OS-9 subdirectories /DEFS, /DEFS/INET */
#else
#include <stdio.h>
#include <types.h>
#include <inet/socket.h>
#include <inet/in.h>
#include <inet/netdb.h>
#include <signal.h>
#include <errno.h>

#endif

#include <time.h>


/* One stream socket is used with adequate throughput                       */
/*   (although two could work, no performance improvement is expected)      */

/* Be careful that you reserve these port numbers to prevent collisions     */
/*    from other processes requesting ports on your system:                 */

#define DISBRIDGE_TCP_PORT     2056 /* disbridge program, server & client   */

                                    /* NPS Autonomous Underwater Vehicle (AUV)*/
                                    /*     Underwater Virtual World (UVW)     */


#define AUVSIM1_TCP_PORT_0     3210 /* os9sender <==> os9server test programs */
#define AUVSIM1_TCP_PORT_1     3211 /* auv execution level <==> virtual world */
#define AUVSIM1_TCP_PORT_2     3212 /*         port for future use            */
#define AUVSIM1_TCP_PORT_3     3213 /*         port for future use            */
#define AUVSIM1_TCP_PORT_4     3214 /*         port for future use            */
#define AUVSIM1_TCP_PORT_5     3215 /*         port for future use            */
#define AUVSIM1_TCP_PORT_6     3216 /*         port for future use            */
#define AUVSIM1_TCP_PORT_7     3217 /*         port for future use            */
```

```
#define AUVSIM1_TCP_PORT_8      3218 /*        port for future use                */
#define AUVSIM1_TCP_PORT_9      3219 /*        port for future use                */

# define TRUE  1
# define FALSE 0

#define SOCKET_QUEUE_SIZE   5       /* max allowed by TCP/IP                     */

/* function prototypes *******************************************************/

/* Note that function prototype parameters may need to be deleted if the
   C compiler used is not an ANSI compiler (e.g. OS-9 K&R C compiler)          */

   void shutdown_os9sender ();

/* global variable definitions ***********************************************/

   static int            TRACE = 0;            /* 1 = trace on, 0 = trace off */

   static int            socket_descriptor,
                         socket_accepted,
                         socket_stream;

   static int            socket_length = 255;

   static int            bytes_received, bytes_read, bytes_written,
                                         bytes_left, bytes_sent;

   int                   shutdown_signal_received = FALSE;

   static int            i            = 0,
                         print_help   = FALSE;

/*****************************************************************************/
/*****************************************************************************/

main (argc, argv)

  int argc; char **argv;     /* command line arguments */
{
   char                     command_sent     [81],
                            command_received [81],
                            remote_host_name [60];

   FILE                    *netstat_fileptr;

   struct sockaddr_in       server_address;

   register struct hostent *server_entity;

   char                     test_buffer [81];

   static char             *ptr;
   static char             *remote_buffer;

/*****************************************************************************/
/* Begin execution, parse command line                                      */

   for (i = 1; i < argc; i++)
   {
      if ((argv[i][0] != '-') || print_help)
      {
```

```
        print_help = TRUE;
        break;
     }
     switch (argv[i][1])    /* switch based on current command line parameter */
     {
        case 'h':       /* help */
        case 'H':       /* Help */
        case '?':       /* Help */
           print_help = TRUE;
           break;

        case 'r':       /* remote hostname tells client who to connect to      */
        case 'R':
           /* get remotehost name */
           if (i+1 < argc) strcpy (remote_host_name, argv[++i]);
           else            print_help = TRUE;
           break;

        case 't':       /* TRACE feature                                       */
        case 'T':
           TRACE = 1;
           break;

        default:        /* no command line entry parameter present      */
           print_help = TRUE;

     } /* end switch based on current command line parameter */
   }    /* end for    */

   if (argc == 1 || print_help) /* print help string *************************/
   {
     printf ("Usage:  os9sender -remote hostname [-help] [-trace]\n");
     exit (-1);
   }
   if (TRACE) printf("[os9sender TRACE on]\n");
   fflush (stdin);

/* Command line parameter parse complete                                       */

/****************************************************************************/
/* Initialize communications blocks                                           */
/****************************************************************************/

/* set socket_length                                                          */

   socket_length = 81;                       /* maximum allowed packet size */


/****************************************************************************/
/* Initialize both client & server *******************************************/

   /* Signal handlers for termination to override net_open () and net_close ()*/
   /*    signal handlers.  Otherwise you are unable to ^C kill this program.  */

#if defined(sgi)
   signal (SIGHUP,  shutdown_os9sender);      /* hangup                        */
   signal (SIGINT,  shutdown_os9sender);      /* interrupt character           */
   signal (SIGKILL, shutdown_os9sender);      /* kill signal from Unix         */
   signal (SIGPIPE, shutdown_os9sender);      /* broken pipe from other host */
   signal (SIGTERM, shutdown_os9sender);      /* software termination          */
#endif

/****************************************************************************/
/* Initialize sender *********************************************************/

   /* start by finding default/desired remote host to connect to              */
   {
```

```
      server_entity = gethostbyname (remote_host_name);
      if (server_entity == NULL)
      {
         printf("os9sender -remote host (\"%s\") not found\n",
                remote_host_name);
         exit (-1);
      }
      else if (TRACE)
           printf("os9sender remote host (\"%s\") located\n", remote_host_name);

      /* Client opens server port ********************************************/

      /* Fill in structure 'server_address' with the address of the         */
      /*         remote host (i.e. SERVER) that we want to connect with:     */

#if defined(sgi)
      bzero ((char *) &server_address, sizeof (server_address));
#endif
      server_address.sin_family   = AF_INET;      /* Internet protocol family */

      /* copy server IP address into sockaddr_in struct server_address       */
#if defined(sgi)
      bcopy (server_entity->h_addr, &(server_address.sin_addr.s_addr),
             server_entity->h_length);
#else
      strncpy(&(server_address.sin_addr.s_addr), server_entity->h_addr,
             server_entity->h_length);
#endif

      /* make sure port is in network byte order                             */
      server_address.sin_port = htons (AUVSIM1_TCP_PORT_0);

      /* Open TCP (Internet stream) socket                                   */
      if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
      {
          printf ("os9sender client can't open server stream socket");
          exit (-1);
      }
      else if (TRACE)
            printf ("os9sender client opened server socket successfully\n");

      /* Connect to the server.  Process will block/sleep until connection is
           is established.  Timeout will return an error.                    */
      if (connect (                 socket_descriptor,
                  (struct sockaddr *) &server_address,
                               sizeof (server_address)) < 0)
      {
          printf ("os9sender client can't connect to server socket\n");
          exit (-1);
      }
      else if (TRACE)
           printf("os9sender client connected to server socket successfully\n");

   } /* end initialization                                                   */

/*****************************************************************************/
/* loop transferring telemetry until shutdown_signal_received:               */
/*****************************************************************************/

   /* Two-way reflector:  listen to local program and relay to remote host,
                          listen to  remote host  and relay to local program  */

   socket_stream = socket_descriptor;   /* client */

   if (TRACE)
   {
     printf ("os9sender CLIENT: socket_descriptor = %d,\n", socket_descriptor);
```

```
    printf ("                       socket_accepted    = %d,\n", socket_accepted);
    printf ("                       socket_stream      = %d\n",  socket_stream);
  }

/***************************************************************************/
/* test handshakes                                                        */

  write (socket_stream, "SUCCESS #2:  os9sender connected to os9server!", 46);
  read  (socket_stream, test_buffer, 46);
  test_buffer [47] = '\n';
  printf ("test handshake between hosts: \n%s\n", test_buffer);

/***************************************************************************/

  /* initialize boolean for main loop control */

  shutdown_signal_received = FALSE;

 while (shutdown_signal_received == FALSE)  /* loop until shutdown.. TRUE    */
 {
  /************************************************************************/
  /* Sender block                                                         */
  /************************************************************************/

  /* read from local stdin, relay to remote host                          */

  gets (command_sent);

  bytes_received = strlen (command_sent);

  if (TRACE) printf ("[os9sender command_sent:%s]\n", command_sent);

  if (bytes_received < 0)  /* read failure                                */
  {
     printf ("os9sender gets () from keyboard unsuccessful\n");
     shutdown_os9sender();
  }

  if (bytes_received > socket_length)
  {
        printf ("os9sender send_telemetry_to_server error: ");
        printf ("bytes_received too big for packet socket_length\n");
        printf ("                ");
        printf ("[bytes_received=%d] > [socket_length=%d]; ",
                     bytes_received,        socket_length);
        printf ("string truncated\n");
  }

  bytes_left        = socket_length;
  bytes_written     = 0;
  ptr               = command_sent;

  while ((bytes_left > 0) && (bytes_written >= 0))  /* write loop ***********/
  {
     bytes_sent = write (socket_stream, ptr, bytes_left);

     if      (bytes_sent <  0) bytes_written = bytes_sent;
     else if (bytes_sent >  0)
         {
            bytes_left     -= bytes_sent;
            bytes_written  += bytes_sent;
            ptr            += bytes_sent;
         }
     if (TRACE)
        printf("os9sender send_telemetry_to_server loop  bytes sent = %d\n",
                bytes_sent);
  }
```

```
if    (bytes_written < 0)
{
      printf ("os9sender send_telemetry_to_server () send failed, ");
      printf ("%d bytes_written\n", bytes_written);
}
else if (TRACE)
     printf ("os9sender send_telemetry_to_server total bytes sent = %d\n",
                    bytes_written);

/*************************************************************************/
/* Check termination                                                      */
/*************************************************************************/

if (strncmp (command_sent, "shutdown", 8) == 0)
    shutdown_signal_received = TRUE;

if (shutdown_signal_received == TRUE)
{
    shutdown_os9sender ();
    break;
}

/*************************************************************************/
/* Receiver block                                                         */
/*************************************************************************/

/* listen to remote host, relay to local network/program                 */

bytes_left          = socket_length;
bytes_received      = 0;
ptr                 = command_received;

while ((bytes_left > 0) && (bytes_received >= 0))  /* read loop ************/
{
   bytes_read = read (socket_stream, ptr, bytes_left);

   if       (bytes_read <  0) bytes_received = bytes_read;
   else if (bytes_read >  0)
   {
        bytes_left     -= bytes_read;
        bytes_received += bytes_read;
        ptr            += bytes_read;
   }
   if (TRACE)
       printf ("os9sender receiver block loop bytes_read = %d\n",
                bytes_read);

   /* if nothing is waiting to be read, break out of read loop            */
   if ((bytes_read == 0) && (bytes_received == 0)) break;
}

if  (bytes_received < 0)      /* failure     */
{
     if (TRACE)
     {
         printf ("os9sender receiver block read failed, ");
         printf ("bytes_received = %d\n", bytes_received);
     }
}
else if (bytes_received == 0)  /* no transfer */
{
     if (TRACE)
         printf("os9sender get_PDU_from_other_host read received 0 bytes\n");
}
else if (bytes_received > 0)  /* success     */

         printf ("%s\n", command_received);
```

```
    /****************************************************************************/
    /* Check termination                                                        */
    /****************************************************************************/

    if (strncmp (command_received, "shutdown", 8) == 0)
        shutdown_signal_received = TRUE;

    if (shutdown_signal_received)
    {
        shutdown_os9sender ();
        break;
    }

} /* end of while (shutdown==FALSE) indefinite loop */

shutdown_os9sender ();         /* ensure shutdown_os9sender is always reached */

printf ("os9sender exit \n");

exit (0);  /* os9sender complete */
}

/******************************************************************************/
/******************************************************************************/
/* end of main program                                                        */
/******************************************************************************/
/******************************************************************************/

/* Shutdown block                                                             */

void shutdown_os9sender ()
{
    if (TRACE) printf ("os9sender shutdown in progress ...\n");

    shutdown_signal_received = TRUE;  /* in case entry was from signal handler */

    /* No need to send a message to other side that bridge is going down,      */
    /*    since SIGPIPE signal trigger may shutdown_os9sender() on other side? */

    if (close (socket_stream) == -1)
        printf ("os9sender close (socket_stream) failed\n");

    if (TRACE) printf ("os9sender shutdown_os9sender () complete\n");
    return;

} /* end shutdown_os9sender () */

/******************************************************************************/
/******************************************************************************/
```

C. *os9server.c* Unix to OS-9 Socket Communications Server

```
/*****************************************************************************
 Program:            os9server.c

 Description:        socket test program to pass input strings in 2 directions,
                        especially socket from OS-9 auvsim1 to Irix auvsim4

 Revised:            20 JAN 94

 Compilation:        unix> cc os9server.c -o os9server
                     os-9> make os9server

 Execution:          host1>   os9server
                     host2>   os9sender -r host1

 Example:            fletch>  os9server
                     auvsim1> os9sender -r fletch


 Execution options:

             -h -t  (first letters are sufficient, capital letters are OK too)

             -help  display calling syntax

            -trace  turn on TRACE mode

 References:      (1) (Gespac-provided) EVIRA EVLAN-11 Ethernet Data Link
                      Controller for the G64/G96 Bus/EVTCP Internet Package
                      technical manuals
                  (2) Internetworking with TCP/IP Volume I:  Principles,
                      Protocols and Architectures, Douglas E. Comer,
                      Prentice Hall, Englewood Cliffs NJ, 1991
                  (3) Internetworking with TCP/IP Volume II:  Design,
                      Implementation and Internals, Douglas E. Comer and
                      David L. Stevens, Prentice Hall, Englewood Cliffs NJ, 1991
                  (4) IRIX Network Programming Guide, Silicon Graphics Inc.
                  (5) An Advanced 4.3BSD Interprocess Communication Tutorial,
                      Samuel J. Leffler, Robert S. Fabry, William N. Joy,
                      Phil Lapsley, Steve Miller and Chris Torek, undated
                  (6) Real-Time Programming Tutorial, Bill Mannel, SGI Expo,
                      Silicon Graphics Inc., 23 May 93
```

```
Author:             Don Brutzman                     brutzman@cs.nps.navy.mil
                    Code OR/Br
                    Naval Postgraduate School        (408) 656-2149 work
                    Monterey CA 93943-5000           (408) 656-2595 fax

Status:             Tested satisfactory Irix<==>Irix, OS-9<==>OS-9, Irix<==>OS-9


Future work:        Consider implementation as an inetd 'superserver' daemon.

                    Ensure compatibility with DIS v2.0 protocols when available.


                    Comments, suggestions and corrections are welcome!


******************************************************************************/
/* Irix defines in /usr/include */
#if defined(sgi)

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <string.h>

/* OS-9 subdirectories /DEFS, /DEFS/INET */
#else
#include <stdio.h>
#include <types.h>
#include <inet/socket.h>
#include <inet/in.h>
#include <inet/netdb.h>
#include <signal.h>
#include <errno.h>

#endif

#include <time.h>


/* One stream socket is used with adequate throughput                        */
/*   (although two could work, no performance improvement is expected)      */

/* Be careful that you reserve these port numbers to prevent collisions      */
/*    from other processes requesting ports on your system:                  */

#define DISBRIDGE_TCP_PORT      2056 /* disbridge program, server & client     */

                                     /* NPS Autonomous Underwater Vehicle (AUV)*/
                                     /*     Underwater Virtual World (UVW)     */


#define AUVSIM1_TCP_PORT_0      3210 /* os9sender <==> os9server test programs */
#define AUVSIM1_TCP_PORT_1      3211 /* auv execution level <==> virtual world */
#define AUVSIM1_TCP_PORT_2      3212 /*         port for future use            */
#define AUVSIM1_TCP_PORT_3      3213 /*         port for future use            */
#define AUVSIM1_TCP_PORT_4      3214 /*         port for future use            */
#define AUVSIM1_TCP_PORT_5      3215 /*         port for future use            */
#define AUVSIM1_TCP_PORT_6      3216 /*         port for future use            */
#define AUVSIM1_TCP_PORT_7      3217 /*         port for future use            */
#define AUVSIM1_TCP_PORT_8      3218 /*         port for future use            */
#define AUVSIM1_TCP_PORT_9      3219 /*         port for future use            */
```

```
# define TRUE  1
# define FALSE 0

#define SOCKET_QUEUE_SIZE   5       /* max allowed by TCP/IP                    */

/* function prototypes *******************************************************/

/* Note that function prototype parameters may need to be deleted if the
   C compiler used is not an ANSI compiler (e.g. OS-9 K&R C compiler)          */

   void shutdown_os9server ();

/* global variable definitions ***********************************************/

   static int            TRACE = 0;             /* 1 = trace on, 0 = trace off */

   static int            socket_descriptor,
                         socket_accepted,
                         socket_stream;

   static int            socket_length = 256;

   static int            bytes_received, bytes_read, bytes_written,
                                         bytes_left, bytes_sent;

   int                   shutdown_signal_received;

   static int            i             = 0,
                         print_help    = FALSE;

/*****************************************************************************/
/*****************************************************************************/

main (argc, argv)

  int argc; char **argv; /* command line arguments */
{
   char                     command_sent     [81],
                            command_received [81],
                            remote_host_name [60];

   FILE                    *netstat_fileptr;

   struct sockaddr_in       server_address;

   register struct hostent *server_entity;

   char                     test_buffer [81];

   static char             *ptr;

/*****************************************************************************/
/* Begin execution, parse command line                                       */

   for (i = 1; i < argc; i++)
   {
      if ((argv[i][0] != '-') || print_help)
      {
         print_help = TRUE;
         break;
      }
      switch (argv[i][1])   /* switch based on current command line parameter */
```

```
    {
      case 'h':      /* help */
      case 'H':      /* Help */
      case '?':      /* Help */
        print_help = TRUE;
        break;

      case 't':      /* TRACE feature                                        */
      case 'T':
        TRACE = 1;
        break;

      default:       /* no command line entry parameter present              */
        break;

    } /* end switch based on current command line parameter */
  }   /* end for    */

  if (print_help)             /* print help string **************************/
  {
    printf ("Usage:  os9server [-help] [-trace]\n");
    exit (-1);
  }
  if (TRACE) printf("[os9server TRACE on]\n");
  fflush (stdin);

/* Command line parameter parse complete                                     */

/****************************************************************************/
/* Initialize communications blocks                                         */
/****************************************************************************/

/* set socket_length                                                         */

  socket_length = 81;                     /* maximum allowed packet size */

/****************************************************************************/
/* Initialize both client & server ******************************************/

  /* Signal handlers for termination to override net_open () and net_close ()*/
  /*    signal handlers.  Otherwise you are unable to ^C kill this program.  */

#if defined(sgi)
  signal (SIGHUP,  shutdown_os9server);     /* hangup                      */
  signal (SIGINT,  shutdown_os9server);     /* interrupt character         */
  signal (SIGKILL, shutdown_os9server);     /* kill signal from Unix       */
  signal (SIGPIPE, shutdown_os9server);     /* broken pipe from other host */
  signal (SIGTERM, shutdown_os9server);     /* software termination        */
#endif

/****************************************************************************/
/* Initialize server ********************************************************/

  /* setup to listen for client to attempt connection                       */
  {
    /* Server opens server port *********************************************/

    /* Open TCP (Internet stream) in socket                                 */
    if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
    {
       printf ("os9server can't 'open' stream socket");
       exit (-1);
    }
    else if (TRACE)
          printf ("os9server socket 'open' successful\n");
```

```
        /* Bind local address so client can talk to server                        */
#if defined(sgi)
        bzero ((char *) &server_address, sizeof (server_address));
#endif
        server_address.sin_family      = AF_INET;   /* Internet protocol family */

        /* make sure port is in network byte order                               */
        server_address.sin_addr.s_addr = htonl (INADDR_ANY);
        server_address.sin_port        = htons (AUVSIM1_TCP_PORT_0);

        if (bind (                  socket_descriptor,
                  (struct sockaddr *) &server_address,
                              sizeof (server_address)) < 0)
        {
            printf("os9server socket 'bind' unsuccessful\n");
            exit (-1);
        }
        else if (TRACE) printf ("os9server socket 'bind' successful\n");

        /* prepare socket queue for connection requests using listen             */

        listen (socket_descriptor, SOCKET_QUEUE_SIZE);
        if (TRACE)
            printf("os9server socket 'listen' successful ...\n");

        /* Server 'accept' waits for client connections ************************/

        if (TRACE) printf ("os9server socket waiting to 'accept' ...\n");
        bytes_received = sizeof (socket_descriptor);
        while ((socket_accepted  = accept ( socket_descriptor,
                                            &server_address,
                                            &bytes_received)) < 1)     /* block */
           if (TRACE)
           {
              printf ("os9server socket 'accept' unsuccessful, ");
              printf ("sleeping 1 second ...\n");
              sleep (1);
           }
        printf("os9server connection is open between networks.\n");

    } /* end initialization                                                    */

/***************************************************************************/
/* loop transferring telemetry until shutdown_signal_received:             */
/***************************************************************************/

    /* Two-way reflector:  listen to local program and relay to remote host,
                           listen to  remote host  and relay to local program  */

    socket_stream = socket_accepted;    /* server */

    if (TRACE)
    {
       printf ("os9server SERVER: socket_descriptor = %d,\n", socket_descriptor);
       printf ("                  socket_accepted   = %d,\n", socket_accepted);
       printf ("                  socket_stream     = %d\n",  socket_stream);
    }

/***************************************************************************/
/* test handshakes                                                         */

    write (socket_stream, "SUCCESS #1:  os9server connected to os9sender!", 46);
    read  (socket_stream, test_buffer, 46);
    test_buffer [47] = '\n';
    printf ("test handshake between hosts: \n%s\n", test_buffer);

/***************************************************************************/
```

```
/* initialize boolean for main loop control */

 shutdown_signal_received = FALSE;

while (shutdown_signal_received == FALSE)     /* loop until shutdown.. TRUE */
{
 /**************************************************************************/
 /* Receiver block                                                         */
 /**************************************************************************/

 /* listen to remote host, relay to local network/program                 */

 bytes_left          = socket_length;
 bytes_received      = 0;
 ptr                 = command_received;

 while ((bytes_left > 0) && (bytes_received >= 0))  /* read loop ***********/
 {
    bytes_read = read (socket_stream, ptr, bytes_left);

    if       (bytes_read <  0) bytes_received = bytes_read;
    else if (bytes_read >  0)
    {
         bytes_left     -= bytes_read;
         bytes_received += bytes_read;
         ptr            += bytes_read;
    }
    if (TRACE)
        printf ("os9server receiver block loop bytes_read = %d\n",
                 bytes_read);

    /* if nothing is waiting to be read, break out of read loop           */
    if ((bytes_read == 0) && (bytes_received == 0)) break;
 }

 if  (bytes_received < 0)       /* failure     */
 {
      if (TRACE)
      {
          printf ("os9server receiver block read failed, ");
          printf ("bytes_received = %d\n", bytes_received);
      }
 }
 else if (bytes_received == 0)  /* no transfer */
 {
      if (TRACE)
          printf("os9server get_PDU_from_other_host read received 0 bytes\n");
 }
 else if (bytes_received > 0)  /* success     */

          printf ("%s\n", command_received);

 /**************************************************************************/
 /* Check termination                                                      */
 /**************************************************************************/

 if (strncmp (command_received, "shutdown", 8) == 0)
     shutdown_signal_received = TRUE;

 if (shutdown_signal_received == TRUE)
 {
     shutdown_os9server ();
     break;
 }

 /**************************************************************************/
```

```
/* Sender block                                                          */
/************************************************************************/

/* read from local stdin, relay to remote host                           */

gets (command_sent);

bytes_received = strlen (command_sent);

if (TRACE) printf ("[os9server command_sent:%s]\n", command_sent);

if (bytes_received < 0)   /* read failure                                */
{
    printf ("os9server gets () from keyboard unsuccessful\n");
    shutdown_os9server ();
}

if (bytes_received > socket_length)
{
        printf ("os9server send_telemetry_to_server error: ");
        printf ("bytes_received too big for packet socket_length\n");
        printf ("              ");
        printf ("[bytes_received=%d] > [socket_length=%d]; ",
                  bytes_received,        socket_length);
        printf ("string truncated\n");
}

bytes_left          = socket_length;
bytes_written       = 0;
ptr                 = command_sent;

while ((bytes_left > 0) && (bytes_written >= 0))   /* write loop ***********/
{
    bytes_sent = write (socket_stream, ptr, bytes_left);

    if       (bytes_sent <  0) bytes_written = bytes_sent;
    else if (bytes_sent >  0)
         {
             bytes_left     -= bytes_sent;
             bytes_written  += bytes_sent;
             ptr            += bytes_sent;
         }
    if (TRACE)
        printf("os9server send_telemetry_to_server loop  bytes sent = %d\n",
                bytes_sent);
}

if   (bytes_written < 0)
{
     printf ("os9server send_telemetry_to_server () send failed, ");
     printf ("%d bytes_written\n", bytes_written);
}
else if (TRACE)
     printf ("os9server send_telemetry_to_server total bytes sent = %d\n",
                     bytes_written);

/************************************************************************/
/* Check termination                                                    */
/************************************************************************/

if (strncmp (command_sent, "shutdown", 8) == 0)
    shutdown_signal_received = TRUE;

if (shutdown_signal_received == TRUE)
{
    shutdown_os9server ();
```

```
        break;
    }

} /* end of while (shutdown_signal_received==FALSE) indefinite loop */

shutdown_os9server ();          /* ensure shutdown_os9server is always reached */

printf ("os9server exit.\n");

exit (0);  /* os9server complete */
}

/****************************************************************************/
/****************************************************************************/
/* end of main program                                                      */
/****************************************************************************/
/****************************************************************************/

/* Shutdown block                                                           */

void shutdown_os9server ()
{
   if (TRACE) printf ("os9server shutdown in progress ...\n");

   shutdown_signal_received = TRUE;  /* in case entry was from signal handler */

   /* No need to send a message to other side that bridge is going down,    */
   /*     since SIGPIPE signal trigger may shutdown_os9sender() on other side? */

   if (close (socket_stream) == -1)
       printf ("os9server close (socket_stream) failed\n");

   if (TRACE) printf ("os9server shutdown_os9server () complete\n");

   return;

} /* end shutdown_os9server () */

/****************************************************************************/
/****************************************************************************/
```

D. *disbridge.c* LAN to LAN Connectivity for DIS without Multicast

```
/*****************************************************************************
 Program:           disbridge.c

 Description:       Relays DIS packets between LANs on different Internet hosts

 Revised:           7 July 93    DIS version 1.3

 Compilation:       % DIS-1.3/src>> make bridge

 Execution:         % host1>> disbridge -server

                    % host2>> disbridge [-client] -remote host1.internet.address

 Execution options:

 -s -c -r -H -I -t  (first letters are sufficient, capital letters are OK too)

           -server  server mode; must be running prior to client calling for it

         [-client]  client mode [default]; remote host must be specified and
                    have disbridge -server already running there

   -remote hostname hostname.remote.net.address for client to connect to server

             -help  display calling syntax

     -interface et0 will force use of a given device interface (e.g. "et0") for
                    multicast packet listening on the local net.  disbridge will
                    attempt to open the first known device if the default device
                    fails.  Only one parent process is allowed to use that
                    device on a given machine.  Thus if you want to connect one
                    local area network (LAN) with another, each LAN must
                    dedicate one machine to disbridge.  This appears to be an
                    unavoidable hardware/multicast restriction.

            -trace  turn on TRACE mode

 References:    (1) Military Standard--Protocol Data Units for Entity
                    Information and Entity Interaction in a
                    Distributed Interactive Simulation (DIS),
                    Institute for Simulation and Training, University of
                    of Central Florida, Orlando FL, 30 OCT 91
                (2) Internetworking with TCP/IP Volume I:  Principles,
                    Protocols and Architectures, Douglas E. Comer,
                    Prentice Hall, Englewood Cliffs NJ, 1991
                (3) Internetworking with TCP/IP Volume II:  Design,
                    Implementation and Internals, Douglas E. Comer and
                    David L. Stevens, Prentice Hall, Englewood Cliffs NJ, 1991
                (4) IRIX Network Programming Guide, Silicon Graphics Inc.
                (5) An Advanced 4.3BSD Interprocess Communication Tutorial,
                    Samuel J. Leffler, Robert S. Fabry, William N. Joy,
                    Phil Lapsley, Steve Miller and Chris Torek, undated
                (6) Real-Time Programming Tutorial, Bill Mannel, SGI Expo,
                    Silicon Graphics Inc., 23 May 93
                (7) DIS v1.2 source library, John Locke, Naval Postgraduate
                    School, Monterey CA, 8 FEB 92
```

```
 Author:              Don Brutzman                    brutzman@cs.nps.navy.mil
                      Code OR/Br
                      Naval Postgraduate School       (408) 656-2149 work
                      Monterey CA 93943-5000          (408) 656-2595 fax

 Acknowledgement:     Thanks to John Locke, Mike Macedonia & Dave Pratt for
assistance.

 Status:              Tested satisfactorily for EntityStatePDUs using v1.2 DIS
protocol.

 Caveat:              Currently transmits EntityStatePDUs after removing
articulated
                         parameters.  This is due to articulated parameter
pointers
                         not surviving transmission of the PDU image over the
socket.

                      Testing to date has been exclusively on SGI wrkstations
running
                      Irix 4.0.5E operating system

 Future work:         Consider implementation as an inetd 'superserver' daemon.

                      Ensure compatibility with DIS v2.0 protocols when available.

                      Consider conversion from iterative to concurrent processing
                         if throughput performance needs improvement.

                      Initialize opposite hosts using ActivateRequest &
                         ActivateReply exchanges, as well as DeActivateRequest &
                         DeActivateReply exchanges.

                      Comments, suggestions and corrections are welcome!


*****************************************************************************/

#include "disdefs.h"   /* DIS Library */

#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <stdio.h>
#include <time.h>
#include <sys/types.h>

/* One stream socket is used with adequate throughput                      */
/*   (although two could work, no performance improvement is expected)     */

/* Be careful that you reserve these port numbers to prevent collisions    */
/*    from other processes requesting ports on your system:                */

#define DISBRIDGE_TCP_PORT     2056 /* disbridge program, server & client   */

                                   /* NPS Autonomous Underwater Vehicle (AUV)*/
                                   /*     Underwater Virtual World (UVW)     */
#define AUVSIM1_TCP_PORT_0     3210 /*                                       */
#define AUVSIM1_TCP_PORT_1     3211 /*             for future use            */
#define AUVSIM1_TCP_PORT_2     3212 /*                                       */
#define AUVSIM1_TCP_PORT_3     3213 /*                                       */
#define AUVSIM1_TCP_PORT_4     3214 /*                                       */
#define AUVSIM1_TCP_PORT_5     3215 /*                                       */
#define AUVSIM1_TCP_PORT_6     3216 /*                                       */
#define AUVSIM1_TCP_PORT_7     3217 /*                                       */
```

```
#define AUVSIM1_TCP_PORT_8      3218 /*                                          */
#define AUVSIM1_TCP_PORT_9      3219 /*                                          */

#define SOCKET_QUEUE_SIZE   5       /* max allowed by TCP/IP                     */

#define SOCKET_PADDING     10       /* a little extra buffer after each PDU      */

/* function prototypes *********************************************************/

/* Note that these function prototypes may need to be commented out it the
   C compiler used is not an ANSI compiler.                                    */

void shutdown_disbridge        (const int status);

void send_PDU_to_other_host    (char *send_PDU, PDUType  send_type);

int  get_PDU_from_other_host   (char **get_PDU, PDUType  *get_type);

void print_PDU                 (char      *pdu, PDUType       type);

/* global variable definitions *************************************************/

   static int             TRACE = 0;             /* 1 = trace on, 0 = trace off */

   static int             socket_descriptor,
                          socket_accepted,
                          socket_stream;

/* maximum allowed socket length value = ETHERMTU [/sys/netinet/ifether.h]     */
/* see initialization code for actual value                                    */
   static int             socket_length = ETHERMTU;

   static int             bytes_received;

   static char           *local_PDU;
   static PDUType         local_PDU_type;

   static char           *remote_PDU;
   static PDUType         remote_PDU_type;

   static int             CLIENT,     /* boolean variables */
                          SERVER;

/* John Locke's carried-over variable definitions *****************************/

   int         i              = 0,
               print_help     = FALSE,
               ethernet_flag  = FALSE,
               nodes,
               read_flag,
               recvd          = 0,
               sent           = 0,
               write_flag;

   extern unsigned short  host_id;

   char          *pdu,
                  ethernet_interface [MAX_INTERF+1];

   PDUType        type;

/******************************************************************************/
/******************************************************************************/
```

```
main (argc, argv)
     int argc; char **argv; /* command line arguments */
{
   char                   command [81],
                          new_device_name [4],
                          remote_host_name [60];

   int                    shutdown;

   FILE                  *netstat_fileptr;

   struct sockaddr_in     server_address;

register struct hostent *server_entity;

   char                   test_buffer [37];

/***************************************************************************/
/* Begin execution, parse command line                                     */

/* Default execution is server waiting for client request to read/write PDUs  */

   SERVER = TRUE;
   CLIENT = FALSE;

   for (i = 1; i < argc; i++)
   {
      if ((argv[i][0] != '-') || print_help)
      {
         print_help = TRUE;
         break;
      }
      switch (argv[i][1])    /* switch based on current command line parameter */
      {
         case 'h':        /* help */
         case 'H':        /* Help */
         case '?':        /* Help */
            print_help = TRUE;
            break;

         case 'i':        /* Ethernet interface */
         case 'I':        /* Ethernet Interface */
            ethernet_flag = TRUE;
            if (i+1 < argc) {
               if (strlen(argv[i+1]) > MAX_INTERF) {
                  printf("%s: error: interface name exceeds %d chars\n",
                                                 argv[0], MAX_INTERF);
                  exit (-1);
               } else {
                  strcpy (ethernet_interface, argv[++i]);
               }
            } else
               print_help = TRUE;
            break;

         case 'c':        /* client initiates server for reading/writing PDUs   */
         case 'C':
            CLIENT = TRUE;
            SERVER = FALSE;
            break;

         case 'r':        /* remote hostname tells client who to connect to     */
         case 'R':
            CLIENT = TRUE;
            SERVER = FALSE;
            /* get remotehost name */
            if (i+1 < argc) strcpy (remote_host_name, argv[++i]);
```

```
            else            print_help = TRUE;
            break;

         case 's':      /* server waits for client request to read/write PDUs */
         case 'S':
            CLIENT = FALSE;
            SERVER = TRUE;
            break;

         case 't':      /* TRACE feature                                      */
         case 'T':
            TRACE = 1;
            break;

         default:
            print_help = TRUE;
      } /* end switch */
   }    /* end for    */

   if (argc == 1 || print_help) /* print help string *************************/
   {
      printf ("Usage:  disbridge [-client -remote hostname | -server <default>]
\n");
      printf ("               [-i device#] [-help] [-trace]\n");
      exit (-1);
   }
   printf ("\n");

   if (!ethernet_flag) strcpy (ethernet_interface, BCAST_INTERF);  /* default */

   if (net_open (ethernet_interface) == FALSE)
   {
      printf ("disbridge net_open (\"%s\") failed,\n", ethernet_interface);
      printf ("   due to improper device number or device is already in use\n");
      net_close ();  /* clean up after a deficient net_open () DIS function   */
                     /* this recovery procedure would be better located there */

      /* Show user what correct port number is                                */
      if (TRACE)
      {
         sprintf (command, "netstat -I");
         printf  (         "netstat -I\n");
         system  (command);           /* display results of netstat -I command */
      }

      sprintf (command, "netstat -I > netstat.I");
      system  (command);

      if ((netstat_fileptr = fopen ("netstat.I", "r")) ==  (FILE *) NULL)
      {
          fprintf ("disbridge failed to create local file netstat.I\n");
          shutdown_disbridge (1);
      }
      /* flush first line then read first device name from file netstat.I     */
      while (command [0] != '\n') fscanf (netstat_fileptr, "%c", &command[0]);
      fscanf  (netstat_fileptr, "%3s", new_device_name);
      fclose  (netstat_fileptr);
      sprintf (command, "rm netstat.I");
      system  (command);

      /* Fixed problem in the DIS library, bad net_close() call caused exit  */
      strcpy (ethernet_interface, new_device_name);
      if (net_open (ethernet_interface) == FALSE)
      {
          printf("disbridge net_open (\"%s\") failed,\n", ethernet_interface);
          printf("   improper device number or device already in use\n");
          exit (-1);
```

```
      }
   }
   printf ("disbridge net_open (\"%s\") succeeded\n", ethernet_interface);

/* Command line parameter parse complete                                     */

/****************************************************************************/
/* Initialize communications blocks                                         */
/****************************************************************************/

/* set socket_length                                                         */

/* socket_length = ETHERMTU;                    */ /* maximum allowed packet size */

   socket_length = sizeof (EntityStatePDU) + SOCKET_PADDING;

/* Note that DIS protocol version 2 will convert 'unused' to PDU length field */

/****************************************************************************/
/* Initialize both client & server *****************************************/

   /* Signal handlers for termination to override net_open () and net_close ()*/
   /*    signal handlers.  Otherwise you are unable to ^C kill this program.  */

   signal (SIGHUP,  shutdown_disbridge);        /* hangup                      */
   signal (SIGINT,  shutdown_disbridge);        /* interrupt character         */
   signal (SIGKILL, shutdown_disbridge);        /* kill signal from Unix       */
   signal (SIGPIPE, shutdown_disbridge);        /* broken pipe from other host */
   signal (SIGTERM, shutdown_disbridge);        /* software termination        */

   /* Future work:  implement inter-network initialization by sending
                    SendActivateRequestPDU's for all entities on the local net
                    to first inform the remote host of their existence       */

   /* not implemented - PDU's copied when received regardless of activation   */

/****************************************************************************/
/* Initialize client *******************************************************/

   if (CLIENT)  /* start by finding remote host to connect to                */
   {
      server_entity = gethostbyname (remote_host_name);
      if (server_entity == NULL)
      {
         printf("disbridge -remote host (\"%s\") not found\n",
                remote_host_name);
         exit (-1);
      }
      else if (TRACE)
           printf("disbridge remote host (\"%s\") located\n", remote_host_name);

      /* Client opens server port *********************************************/

      /* Fill in structure 'server_address' with the address of the
         remote host (i.e. SERVER) that we want to connect with              */

      bzero ((char *) &server_address, sizeof (server_address));
      server_address.sin_family   = AF_INET;       /* Internet protocol family */

      /* copy server IP address into sockaddr_in struct server_address       */
      bcopy (server_entity->h_addr, &(server_address.sin_addr.s_addr),
             server_entity->h_length);

      /* make sure port is in network byte order                             */
      server_address.sin_port = htons (DISBRIDGE_TCP_PORT);
```

```
      /* Open TCP (Internet stream) socket                                      */
      if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
      {
         printf ("disbridge client can't open server stream socket");
         exit (-1);
      }
      else if (TRACE)
            printf ("disbridge client open server socket successful\n");

      /* Connect to the server.  Process will block/sleep until connection is
            is established.  Timeout will return an error.                      */
      if (connect (                  socket_descriptor,
                  (struct sockaddr *) &server_address,
                              sizeof (server_address)) < 0)
      {
         printf ("disbridge client can't connect to server socket\n");
         exit (-1);
      }
      else if (TRACE)
           printf("disbridge client connect to server socket successful\n");

   } /* end if (CLIENT) initialization                                          */

/***************************************************************************/
/* Initialize server *******************************************************/

   if (SERVER)  /* setup to listen for client to attempt connection            */
   {
      /* Server opens server port *******************************************/

      /* Open TCP (Internet stream) in socket                                   */
      if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
      {
         printf ("disbridge server can't 'open' stream socket");
         exit (-1);
      }
      else if (TRACE)
             printf ("disbridge server socket 'open' successful\n");

      /* Bind local address so client can talk to server                       */
      bzero ((char *) &server_address, sizeof (server_address));
      server_address.sin_family      = AF_INET;   /* Internet protocol family */
      server_address.sin_addr.s_addr = htonl (INADDR_ANY);
      server_address.sin_port        = htons (DISBRIDGE_TCP_PORT);

      if (bind (                  socket_descriptor,
               (struct sockaddr *) &server_address,
                           sizeof (server_address)) < 0)
      {
         printf("disbridge server socket 'bind' unsuccessful\n");
         exit (-1);
      }
      else if (TRACE) printf ("disbridge server socket 'bind' successful\n");

      /* prepare socket queue for connection requests using listen              */

      listen (socket_descriptor, SOCKET_QUEUE_SIZE);
      if (TRACE)
         printf("disbridge server socket 'listen' successful ...\n");

      /* Server 'accept' waits for client connections *************************/

      if (TRACE) printf ("disbridge server socket waiting to 'accept' ...\n");
      bytes_received = sizeof (socket_descriptor);
      while ((socket_accepted  = accept ( socket_descriptor,
                                          &server_address,
                                          &bytes_received)) < 1)     /* block */
```

```
        if (TRACE)
        {
           printf ("disbridge server socket 'accept' unsuccessful, ");
           printf ("sleeping 1 second ...\n");
           sleep (1);
        }
     printf("disbridge connection is open between networks.\n");

  } /* end if (SERVER) initialization                                        */

/***************************************************************************/
/* loop until shutdown:                                                    */
/***************************************************************************/

  /* Two-way reflector:  listen to local network and relay to remote host,
                          listen to  remote host  and relay to local network  */

  if      (SERVER)     socket_stream = socket_accepted;
  else /* CLIENT */    socket_stream = socket_descriptor;

  if (TRACE)
  {
     if (CLIENT) printf ("disbridge CLIENT:  ");
     if (SERVER) printf ("disbridge SERVER:  ");
     printf(              "socket_descriptor = %d,\n", socket_descriptor);
     printf("              socket_accepted   = %d,\n", socket_accepted);
     printf("              socket_stream     = %d\n",  socket_stream);
  }

/***************************************************************************/
/* test handshakes                                                         */

  if (CLIENT)
      write (socket_stream, "disbridge CLIENT connected to SERVER", 36);
  if (SERVER)
      write (socket_stream, "disbridge SERVER connected to CLIENT", 36);
  read (socket_stream, test_buffer, 36);
  test_buffer [37] = '\n';
  printf ("test handshake between hosts:  %s\n", test_buffer);

/***************************************************************************/

  /* After this point the behaviors of the CLIENT and SERVER are identical
       so we no longer bother testing for which version is being executed   */

  shutdown = FALSE;               /* initialize boolean for main loop control */

 while (shutdown == FALSE)    /* begin to loop indefinitely until shutdown met */
 {
  /*************************************************************************/
  /* Sender block                                                          */
  /*************************************************************************/

  /* read local_PDU from local network, relay it to remote host             */

  nodes = net_read (&local_PDU, &local_PDU_type);     /* note pointers passed */

  if (nodes < 0)  /* net_read failure                                       */
  {
     printf ("disbridge net_read () of local PDU traffic unsuccessful\n");
     shutdown_disbridge (1);
  }
  else if (nodes > 0)  /* at least one PDU found, print it if in TRACE mode  */
  {
    if (TRACE)
    {
       printf ("disbridge net_read () local_PDU captured successfully:\n");
```

```
            print_PDU (local_PDU, local_PDU_type);
        }

        /* PDU found.  No need to check origin tag prior to relaying it since
           DIS library will ensure that net_read () does not see PDUs that
           originated from the self process's net_write ().  Such checks are
           essential at some level, or a pathological race condition will develop
           and possibly swamp both networks with duplicated PDU traffic.          */

        send_PDU_to_other_host (local_PDU, local_PDU_type);
    }
    else /* nodes == 0, no pdu found, no action required except trace          */
         if (TRACE)
             printf ("disbridge net_read () no local PDU traffic pending ...\n");

    /***********************************************************************/
    /* Receiver block                                                      */
    /***********************************************************************/

    /* listen to remote host, relay remote PDU to local network            */

    /* if remote_PDU found, net_write it                                   */

    if (get_PDU_from_other_host (&remote_PDU, &remote_PDU_type) > 0)
    {
        if   (TRACE)
        {
              printf ("disbridge get_PDU_from_other_host successful:\n");
              print_PDU (remote_PDU, remote_PDU_type);
              printf ("disbridge calling net_write () ");
              printf ("to issue remote_PDU locally ...\n");
        }
        if   (net_write (remote_PDU, remote_PDU_type) == FALSE)
              printf ("disbridge net_write (remote_PDU) unsuccessful\n");

        else  if (TRACE)
              printf ("disbridge net_write (remote_PDU) successful\n");
    }
    else if (TRACE) printf ("disbridge get_PDU_from_other_host unsuccessful\n");

    /***********************************************************************/
    /* Check termination                                                   */
    /***********************************************************************/

    if (shutdown) shutdown_disbridge (0);

    if (TRACE)
    {
        printf ("disbridge no PDUs pending, sleep (1) ...\n");
        sleep (1);
    }
    /* shutdown = TRUE; */

  } /* end of while (shutdown==FALSE) indefinite loop */

  shutdown_disbridge (0);  /* ensure shutdown_disbridge is always reached      */

  exit (0);  /* disbridge complete */
}
/*************************************************************************/
/*************************************************************************/
/* end of main program                                                   */
/*************************************************************************/
/*************************************************************************/


/*************************************************************************/
```

```
/* Shutdown block                                                          */
/***************************************************************************/

void shutdown_disbridge (status)

               const int status;
{
   if (TRACE) printf ("disbridge shutdown in progress ...\n");

   /* No need to send a message to other side that bridge is going down,
         since SIGPIPE signal triggers shutdown_disbridge () the other side   */

   if (close (socket_stream) == -1)
       printf ("disbridge close (socket_stream) failed\n");

   net_close (); /* close local net port connection listening for DIS PDU's   */
                 /* watch out for net_close side effects/interrupts disabled! */

   printf ("disbridge exit.\n");
   exit (status);  /* status parameter indicates normal or abnormal exit      */

} /* end shutdown_disbridge () */

/***************************************************************************/

void send_PDU_to_other_host  (send_PDU, send_PDU_type)

                              char     *send_PDU;
                              PDUType   send_PDU_type;
{
   int             i, bytes_written, bytes_left, bytes_sent;
   char           *ptr;
   EntityStatePDU *cast_PDU;

   char PDU_character_buffer [ETHERMTU];  /* ETHERMTU: /sys/netinet/ifether.h */

   if (send_PDU_type == EntityStatePDU_Type)
   {
       /* prevent passing articulated parameter ponters
*/
       cast_PDU = (EntityStatePDU *) send_PDU;
       cast_PDU->num_articulat_params = 0;

       bcopy (send_PDU, PDU_character_buffer, socket_length);
       if (TRACE)
       {
           pRintf ("disbridge send_PDU_to_other_host socket_length = %d\n",
                    socket_length);
           printf ("disbridge send_PDU type:  %d\n", send_PDU_type);
           printf ("disbridge send_PDU character_buffer = [");
           for (i=0; i < socket_length; i++)
                printf ("%x ", PDU_character_buffer [i]);
           printf ("]\n");
           printf ("disbridge send_PDU character_buffer = [");
           for (i=0; i < socket_length; i++)
                printf ("%c", PDU_character_buffer [i]);
           printf ("]\n"); fflush (stdout);
       }

       if (socket_length > ETHERMTU)
       {
           printf ("disbridge send_PDU_to_other_host error: ");
           printf ("PDU too big for packet");
           printf ("          ");
           printf ("[PDU socket_length=%d] > [ETHERMTU=%d]; nothing sent\n",
                          socket_length,       ETHERMTU);
           return;
```

```
        }

        bytes_left          = socket_length;
        bytes_written       = 0;
        ptr                 = PDU_character_buffer;

    while ((bytes_left > 0) && (bytes_written >= 0))            /* write loop */
    {
        bytes_sent = write (socket_stream, ptr, bytes_left);

        if      (bytes_sent <  0) bytes_written = bytes_sent;
        else if (bytes_sent >  0)
        {
            bytes_left     -= bytes_sent;
            bytes_written  += bytes_sent;
            ptr            += bytes_sent;
        }
        if (TRACE)
            printf("disbridge send_PDU_to_other_host loop bytes sent = %d\n",
                    bytes_sent);
    }
        if   (bytes_written <= 0)
        {
             printf ("disbridge send_PDU_to_other_host () send failed, ");
             printf ("%d bytes_written\n", bytes_written);
        }
        else   if (TRACE)
         printf ("disbridge send_PDU_to_other_host total bytes_written = %d\n",
                         bytes_written);
    }
    else printf ("disbridge unsupported PDUType (%d), not passed to remote\n",
                 send_PDU_type);

} /* end send_PDU_to_other_host () */

/***************************************************************************/

int get_PDU_from_other_host (get_PDU, get_PDU_type) /* returns 1 if PDU found */

                             char   **get_PDU;
                             PDUType *get_PDU_type;
{
        int                 bytes_read, bytes_left;
        char               *ptr;

   static char             *remote_PDU_buffer;
   static PDUHeader        *remote_header;

   remote_PDU_buffer = malloc (socket_length);
   if (remote_PDU_buffer == NULL)
   {
       printf
         ("disbridge get_PDU_from_other_host remote_PDU_buffer malloc error\n");
       return (0);    /* no PDU returned */
   }
   bytes_left         = socket_length;
   bytes_received     = 0;
   ptr                = remote_PDU_buffer;

   while ((bytes_left > 0) && (bytes_received >= 0))            /* read loop */
   {
      bytes_read = read (socket_stream, ptr, bytes_left);

      if      (bytes_read <  0) bytes_received = bytes_read;
      else if (bytes_read >  0)
      {
           bytes_left     -= bytes_read;
```

```
        bytes_received += bytes_read;
        ptr            += bytes_read;
      }
      if (TRACE)
          printf ("disbridge get_PDU_from_other_host loop bytes_read = %d\n",
                  bytes_read);

      /* if nothing is waiting to be read, break out of read loop          */
      if ((bytes_read == 0) && (bytes_received == 0)) break;
   }

   if  (bytes_received < 0)       /* failure      */
   {
       if (TRACE)
       {
           printf ("disbridge get_PDU_from_other_host read failed, ");
           printf ("bytes_received = %d\n", bytes_received);
       }
       return (0);  /* no PDU returned */
   }
   else if (bytes_received == 0)  /* no transfer */
   {
       if (TRACE)
           printf("disbridge get_PDU_from_other_host read received 0 bytes\n");
       return (0);  /* no PDU returned */
   }
   /* else (bytes_received > 0)  /* success      */

   /* Cast and convert buffer to return remote_PDU and remote_PDY_type       */

   remote_header  = (PDUHeader *) remote_PDU_buffer;
   *get_PDU       =               remote_PDU_buffer;
   *get_PDU_type  = (PDUType)     remote_header->type;

   if (TRACE)
   {
       printf("disbridge get_PDU_from_other_host read bytes_received = %3d \n",
               bytes_received);
       printf("disbridge get_PDU_from_other_host read socket_length = %3d \n",
               socket_length);
       printf ("disbridge remote_header get_PDU_type:  %d\n", *get_PDU_type);
       printf ("disbridge get_PDU  remote_PDU_buffer = [");
       for (i=0; i < socket_length; i++)
           printf ("%x ", remote_PDU_buffer [i]);
       printf ("]\n");
       printf ("disbridge get_PDU () remote_PDU_buffer = [");
       for (i=0; i < socket_length; i++)
           printf ("%c", remote_PDU_buffer [i]);
       printf ("]\n"); fflush (stdout);
       ptr = *get_PDU;
       printf ("disbridge get_PDU () get_PDU           = [");
       for (i=0; i < socket_length; i++)
           printf ("%c", ptr [i]);
       printf ("]\n"); fflush (stdout);
   }
   return (1);  /* PDU returned */

}     /* end get_PDU_from_other_host () */

/****************************************************************************/

void print_PDU (pdu, type)               /* from John Locke's test_it.c     */
           char *pdu;  PDUType type;     /* modified to stand as a routine, */
                                         /* no other changes made           */
{
   /* example PDUs        */
   EntityStatePDU        *ESpdu;
```

```
FirePDU                 *Fpdu;
DetonationPDU           *Dpdu;
ServiceRequestPDU       *SRpdu;
ResupplyPDU             *Rpdu;
ResupplyCancelPDU       *RCpdu;
RepairCompletePDU       *RC_pdu;
RepairResponsePDU       *RRpdu;
CollisionPDU                    *Cpdu;
ActivatRequestPDU       *ARpdu;
ActivatResponsePDU      *AR_pdu;
DeactivatRequestPDU     *DRpdu;
DeactivatResponsePDU            *DR_pdu;
EmitterPDU              *Epdu;
RadarPDU                *Rad_pdu;

ArticulatParamsNode     *APNptr;                /* list nodes */
SupplyQtyNode           *SQNptr;
RadarSystem             *rad_sys_ptr;
RadarSystemNode         *RSN_ptr;
RadarSystemsNode        *RSNptr;
StoresNode              *SNptr;
EmitterNode             *ENptr;
IlluminedEntityNode     *IENptr;

      /* Print PDU data received */
      if (TRACE) printf("disbridge print_PDU () PDU Type = %d\n", type);
      switch (type)
      {
      case (OtherPDU_Type):
         printf("Undefined PDU type (OtherPDU_Type) received.\n");
         break;
      case (FirePDU_Type):
         printf("Got %d:FirePDU\n", FirePDU_Type);
         Fpdu = (FirePDU *) pdu;
         printf("site: %d\n", Fpdu->firing_entity_id.address.site);
         printf("range: %f\n", Fpdu->range);
         break;
      case (EntityStatePDU_Type):
         printPDU((char *) pdu);
         break;
      case (DetonationPDU_Type):
         printf("Got %d:DetonationPDU\n", DetonationPDU_Type);
         Dpdu = (DetonationPDU *) pdu;
         break;
      case (ServiceRequestPDU_Type):
         printf("Got %d:ServiceRequestPDU\n", ServiceRequestPDU_Type);
         SRpdu = (ServiceRequestPDU *) pdu;
         printf("quantity = %f\n",
                    SRpdu->supply_qty_head->supply_quantity.quantity);
         break;
      case (ResupplyOfferPDU_Type):
         printf("Got %d:ResupplyOfferPDU\n", ResupplyOfferPDU_Type);
         Rpdu = (ResupplyPDU *) pdu;
         printf("offer quantity = %f\n",
                    Rpdu->supply_qty_head->supply_quantity.quantity);
         break;
      case (ResupplyReceivedPDU_Type):
         printf("Got %d:ResupplyReceivedPDU\n", ResupplyReceivedPDU_Type);
         Rpdu = (ResupplyPDU *) pdu;
         printf("received quantity = %f\n",
                    Rpdu->supply_qty_head->supply_quantity.quantity);
         break;
      case (ResupplyCancelPDU_Type):
         printf("Got %d:ResupplyCancelPDU\n", ResupplyCancelPDU_Type);
         break;
      case (RepairCompletePDU_Type):
         printf("Got %d:RepairCompletePDU\n", RepairCompletePDU_Type);
```

```
            break;
         case (RepairResponsePDU_Type):
            printf("Got %d:RepairResponsePDU\n", RepairResponsePDU_Type);
            break;
         case (CollisionPDU_Type):
            printf("Got %d:CollisionPDU\n", CollisionPDU_Type);
            break;

         /* Experimental PDU Types */
         case (ActivatRequestPDU_Type):
            printf("Got %d:ActivatRequestPDU\n", ActivatRequestPDU_Type);
            ARpdu = (ActivatRequestPDU *) pdu;
            printf("radar_system = %d\n",
                      ARpdu->radar_systems_head->radar_systems.radar_system);
            printf("stores qty = %f\n", ARpdu->stores_head->stores.quantity);
            printf("ID = %d\n",
                         ARpdu->articulat_params_head->articulat_params.ID);
            break;
         case (ActivatResponsePDU_Type):
            printf("Got %d:ActivatResponsePDU\n", ActivatResponsePDU_Type);
            break;
         case (DeactivatRequestPDU_Type):
            printf("Got %d:DeactivatRequestPDU\n", DeactivatRequestPDU_Type);
            break;
         case (DeactivatResponsePDU_Type):
            printf("Got %d:DeactivatResponsePDU\n", DeactivatResponsePDU_Type);
            break;
         case (EmitterPDU_Type):
            printf("Got %d:EmitterPDU\n", EmitterPDU_Type);
            Epdu = (EmitterPDU *) pdu;
            printf("entity_type = %d\n", Epdu->entity_type);
            printf("num_emitters = %d\n", Epdu->num_emitters);
            printf("emitter_params.param_1 = %d\n",
                         Epdu->emitter_head->emitter.emitter_params.param_1);
            break;
         case (RadarPDU_Type):
            printf("Got %d:RadarPDU\n", RadarPDU_Type);
            Rad_pdu = (RadarPDU *) pdu;
            printf("location_to_entity.x = %f\n",
                Rad_pdu->radar_system_head->radar_system.location_to_entity.x);
            rad_sys_ptr = &(Rad_pdu->radar_system_head->radar_system);
            printf("target_id.address.site = %X\n",
                     rad_sys_ptr->illumined_entity_head->illumined_entity.
                                                 target_id.address.site);
            break;

         default:
            printf("Invalid PDU type received.\n");
            break;

         }
} /* end print_PDU () */

/*****************************************************************************/
/*****************************************************************************/
```

VIII. WORLD-WIDE WEB (WWW) HYPERMEDIA AND MULTICAST BACKBONE (MBone)

A. Introduction

The World-Wide Web (WWW) project has been defined as a "wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents" (Hughes 94). Fundamentally the WWW combines a name space consisting of any information store available on the Internet with a broad set of retrieval clients and servers, all of which can be connected by easily-defined hypertext markup language (*.html*) multimedia links. This globally-accessible combination of media, client programs, servers and hyperlinks can be conveniently utilized by humans or autonomous entities. The Web has fundamentally shifted the nature of information storage, access and retrieval (Hughes 94) (Berners-Lee 94a, 94b).

Universal Resource Locators (URLs) are a key WWW innovation. A block of information might contain text, document, image, sound clip, video clip, executable program, archived dataset or arbitrary stream. If that block of information exists on the Internet, it can be uniquely identified by host machine IP address, publicly visible local directory, local file name, and type of client needed for retrieval (such as anonymous ftp, hypertext browser or gopher). Ordinarily the local file name also includes an extension which identifies the media type (such as *.ps* for PostScript file or *.rgb* for an image). Thus the URL completely specifies everything needed to retrieve any type of electronic information resource. Example URLs appear in the list of references, e.g. (Hughes 94).

The Multicast Backbone (MBone) permits several-to-many simultaneous high-bandwidth communications streams across the Internet. Detailed information on MBone applications and use appears in (Macedonia, Brutzman 94). A customized session directory (*sd*) configuration file *.sd.tcl* is provided. This configuration file adjusts the *sd* interface at run time to enable automatic DIS connection of the

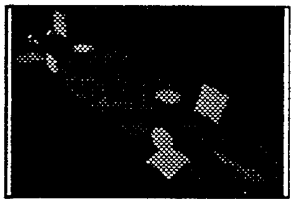
underwater virtual world *viewer* and automatic spawning of a *Mosaic* browser connection to the NPS AUV WWW Home Page. Thus, once a system has been configured using free software for *Mosaic*, MBone and this project, connection to an already-running underwater virtual world requires only a single button click from any compatible workstation on the Internet. Global connectivity and ease of use are thus demonstrated in order to further encourage remote collaborative research and large-scale widely-distributed virtual worlds.

B. NPS AUV World-Wide Web Home Page

Hot links to other home pages, servers or media are portrayed by underlined text in the home page. Line breaks and page layout are handled automatically by the browser according to window and font size.

NPS Autonomous Underwater Vehicle (AUV) World-Wide Web Home Page
ftp://taurus.cs.nps.navy.mil/pub/auv/auv.html

NPS Autonomous Underwater Vehicle (AUV)

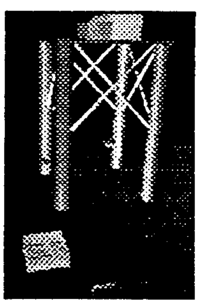


(about Say...)

NPS AUV publication abstracts and anonymous ftp server

NPS AUV graphics rendering and a wireframe rendering

NPS AUV Underwater Virtual World



finger auv@dude.cs.nps.navy.mil for a sample mission report.

A free tar archive of the virtual world software is here: *auv-uvw.tar.Z*, including source code and executable binaries that run on Silicon Graphics (SGI) workstations.

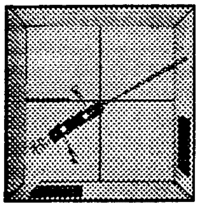
A user installation guide is available. Please let me know if you need help installing the virtual world. Research collaboration is welcome.

Here is another sample NPS AUV mission report run in the virtual world.

Additional supporting papers and programs used with the virtual world software are distributed separately:

- *"From virtual world to reality: designing an autonomous underwater robot"* paper and slides.
- **Vehicle telemetry** postscript plots (20 pages) for the SIGGRAPH mission.
- Mission script syntax for NPS AUV execution level control: mission.script.HELP file.

- **www** line mode World-Wide Web browser distribution page for the **www** anonymous ftp directory. **www** is used to make World-Wide Web queries from inside the virtual world. This program is needed if you want live text to speech capability. Put **www** in the /dynamics directory or have your system administrator install it.
- **gnuplot** plotting program distribution directory (and FAQ page). **gnuplot** is used to plot robot telemetry results. This program is needed if you want a plotting capability. Put it in the /execution directory or have your system administrator install it.
- **MBone** Multicast Backbone connection information. An MBone connection is needed if you want to participate in worldwide audio/video/DIS multicasts with the underwater virtual world. If you are a local user on the gravy cs.nps.navy.mil subnet, you don't have to download the full distribution but instead can copy the MBone session directory (**sd**) configuration file **.sd.tcl** and **.mailcap** mosaic initialization file to your root directory, and then paste the following **.cshrc** aliases into your root directory **.cshrc** file. After you **source .cshrc** you are ready to run **sd** and **mosaic**.



The **NPS AUV Underwater Virtual World** also appeared at **The Edge** exhibition at **SIGGRAPH 94**, July 26-29 in Orlando Florida USA. There was a simultaneous MBone multicast on the worldwide Internet all that week. Information resources from that exhibit include:

- project abstract,
- collaborator list,
- people pages,
- 1000-word description and proposal for exhibition in
- **The Edge** exhibition at **SIGGRAPH 94**, and
- information regarding the audio/video multicast on the Internet MBone.

The Universal Resource Locator (URL) for this home page is
ftp://taurus.cs.nps.navy.mil/pub/auv/auv.html

C. Hypertext Markup Language (HTML) Syntax Summary

The following HTML syntax summary will help decipher most of the home page document which follows in the next section. The key feature of HTML is that it describes document structure: images, the relative emphasis of text entries and links to other documents or files via URLs (NCSA 94b).

The rendered format of an HTML document can vary depending on the browser used to access it. This permits both graphics terminals and character-based terminals to each access and display a document, preserving contextual links throughout. URL media format is indicated by the filename extension. The ability to display movies, play sounds, and utilize other media depends on the capabilities of the connecting machine, the presence of 'viewer' programs which can receive and output the media after a handoff from the browser, and proper links between MIME types and browsers as configured in the user's *.mailcap* file.

The following is a short synopsis of key HTML markups. Markup tags are generally case insensitive. Blanks and skipped lines between markup tags have no effect.

See URL *http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html* for detailed information on HTML syntax.

<B> **bold text** </B>

<I> *italics text* </I>

<P> inserts a paragraph break.

<HR> places a horizontal rule before the next item:

<A IMG="*universal resource locator*">
places an inline image on the page.

<A HREF="*universal resource locator*"> button text>
places button text on the screen which retrieves "*universal resource locator*" when selected. Inline images can be used in place of text.

 starts an unnumbered list

 starts a list item

 starts another list item

 ends an unnumbered list

Figure 5. Hypertext Markup Language (HTML) Syntax Summary.

D. *auv.html* NPS AUV Home Page (Hypertext Markup Language)

```
<HEADER>
<TITLE> NPS Autonomous Underwater Vehicle (AUV) World-Wide Web Home Page</TITLE>
</HEADER>

  <H1>      <A NAME=AUV> NPS Autonomous Underwater Vehicle (AUV) </H1>

          <A HREF="ftp://taurus.cs.nps.navy.mil/pub/auv/auv.iv.gif"><IMG
SRC="ftp://taurus.cs.nps.navy.mil/pub/auv/auv.iv.gif" ALT="NPS AUV logo"></A>
          <A HREF="ftp://taurus.cs.nps.navy.mil/pub/auv/nps_auv.au"><IMG
SRC="ftp://taurus.cs.nps.navy.mil/pub/mosaic/sound.xbm" ALT="sound icon"></A>.
          <A
HREF="http://www_tios.cs.utwente.nl:8001/say/?Naval+Postgraduate+School,Autonomous+Und
erwater+Vehicle"><IMG SRC="ftp://taurus.cs.nps.navy.mil/pub/mosaic/sound.xbm"
ALT="sound icon"></A>
          (about <A HREF = "http://www_tios.cs.utwente.nl/say/?"> Say...</A>)

  <H2>      NPS AUV <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/AUVPAPERS">
publication abstracts</A> and

                  <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv"> anonymous ftp
server<i>(ftp://taurus.cs.nps.navy.mil/pub/auv)</i></A>  </H2>

      <B> NPS AUV </B> <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv.iv.rgb.Z">
graphics rendering</A> and a
                        <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/auv.iv.wireframe.rgb.Z"> wireframe rendering</A>
<P>

<hr>

  <H1>      <A NAME="AUV-UVW"> NPS AUV Underwater Virtual World</H1>

          <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.gif"><IMG SRC =
"ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.gif" ALT="AUV Underwater Virtual World
image"></A> <P>

          Finger the NPS AUV account <A HREF =
"http://www.cs.indiana.edu/finger/gateway?auv@dude.cs.nps.navy.mil"><i>(finger
auv@dude.cs.nps.navy.mil)</i></A> for a sample mission report.<P>

    <B>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.tar.Z">A free tar
archive of the virtual world software is here: auv-uvw.tar.Z</A> </B>
          including source code and executable binaries that run on Silicon Graphics
(SGI) workstations.<P>

          A <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.virtual-world.INSTALL">user installation
guide</A>
          is available.  Please let me know if you need help installing the virtual
world.  Research collaboration is welcome. <P>

          Here is another sample <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/auv-uvw.virtual-world.README">NPS AUV mission
report</A> run in the virtual world. <P>

          Additional supporting papers and programs used with the virtual world
softare are distributed separately:
    <ul>
    <li>    <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/aaai92ws.ps.Z"><i>"From
virtual world to reality: designing an autonomous underwater robot"</i> paper</A> and
          <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/aaai92ws.slides.ps.Z">slides</A>.
```

```
    <li>    <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/AUV_telemetry.ps.Z"><B>Vehicle telemetry</B>
postscript plots (20 pages) for the SIGGRAPH mission</A>.

    <li>    <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/mission.script.HELP">
Mission script syntax for NPS AUV execution level control</A>: mission.script.HELP
file.


    <li>    <A HREF =
"http://info.cern.ch/hypertext/WWW/LineMode/Defaults/Distribution.html"><B>www</B>
line mode World-Wide Web browser</A> distribution page for the
            <A HREF = "ftp://info.cern.ch/pub/www/src/"><B>www</B> anonymous ftp
directory</A>.
            <b>www</b> is used to make World-Wide Web queries from inside the virtual
world.  This program is needed if you want live text to speech capability.
            Put <b>www</b> in the /dynamics directory or have your system administrator
install it.

    <li>    <A HREF = "http://www.cs.dartmouth.edu/gnuplot/">
<B>gnuplot</B> plotting program</A> distribution directory (and
            <A HREF = "ftp://ftp.dartmouth.edu/pub/gnuplot/faq/gpt_faq.html"> FAQ
page</A>).
            <b>gnuplot</b> is used to plot robot telemetry results.  This program is
needed if you want a plotting capability.
            Put it in the /execution directory or have your system administrator
install it.

    <li>    <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/mosaic/mbone.html"><B>MBone</B>
Multicast Backbone</A> connection information.
            An MBone connection is needed if you want to participate in worldwide
audio/video/DIS multicasts with the underwater virtual world.  <P>
            If you are a local user on the gravy1.cs.nps.navy.mil subnet, you don't
have to download the full distribution but instead can copy the MBone session director
(<b>sd</b>) configuration file
            <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/.sd.tcl">
<B>.sd.tcl</B></A> and
            <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/.mailcap"><B>.mailcap</B>
mosaic initialization file</A> to your root directory, and then paste the following
            <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/.cshrc-excerpt"><B>.cshrc</B> aliases</A> into
your root directory <B>.cshrc</B> file.   After you <B>source .cshrc</b> you are ready
to run <b>sd</b> and <b>mosaic</b>.<P>

   </ul>

            <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/pool-search.gif"> <IMG SRC
= "ftp://taurus.cs.nps.navy.mil/pub/auv/pool-search.gif" ALT="AUV pool search
image"></A><P>

            The <B>NPS AUV Underwater Virtual World</B> also appeared at
            <A HREF = "http://www.vsl.ist.ucf.edu/projects/edge/edge.html">
     <B>The Edge</B></A> exhibition at
            <A HREF = "gopher://siggraph.org/00/conferences/siggraph94/gip/gip.txt">
     <B> SIGGRAPH 94</B></A>, July 26-29 in Orlando Florida USA.
            There was a simultaneous
            <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/SIGGRAPH94-mbone.html"> MBone
multicast on the worldwide Internet</A> all that week.
            Information resources from that exhibit include:
   <ul>
   <li>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/abstract.txt">
     project abstract</A>,
   <li>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/contacts.txt">
     collaborator list</A>,
   <li>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/pratts/home.html">
     people pages</A>,
   <li>     <A HREF =
```

```
"ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/edge_description.txt"> 1000-word
description</A> and
          <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/edge_proposal.ps.Z">   proposal</A>
for exhibition in
   <li>     <A HREF = "http://www.vsl.ist.ucf.edu/projects/edge/edge.html">
     <B>The Edge</B></A> exhibition at
          <A HREF = "gopher://siggraph.org/00/conferences/siggraph94/gip/gip.txt">
     <B> SIGGRAPH 94</B></A>, and
   <li>     information regarding the <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/SIGGRAPH94/SIGGRAPH94-mbone.html">audio/video
multicast on the Internet MBone</A>.
 </ul>

<hr>

     <H1>    IEEE Oceanic Engineering Society AUV 94 Conference</H1>

        <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv94.logo.gif"> <IMG SRC =
"ftp://taurus.cs.nps.navy.mil/pub/auv/auv94.logo.gif" ALT="AUV 94 logo"></A> <P>

          AUV 94 took place July 19-20 1994 in Cambridge Massachusetts USA.
   <ul>
   <li>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv94.program.txt">
Conference Program in text</A> and
          <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv94.ps.Z">
postscript</A> formats,
   <li>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv94video.info">
Video Proceedings & broadcast information</A>, and the
   <li>     <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/auv94video.ps.Z">
Video Proceedings abstract booklet</A>.
   </ul>
          We were able to broadcast the video proceedings and selected sessions over
the MBone.
          Conference e-mail queries may be sent to <i>auv94@ieee.org</i>  <P>

<hr>
     <H2>  Other good stuff! </H2>

          <A HREF = "ftp://ftp.nps.navy.mil/pub/usw/av_mcm.html"> <B>Symposium on
Autonomous Systems in Mine Countermeasures</B></A> at NPS April 4-6 1995.  <P>

          Unmanned Untethered Submersible Technology (UUST) 93 <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/uust93_video_order.form"> video proceedings
abstracts</A> with order form <P>

          <A HREF = "http://www.cs.wisc.edu/~pruyne/os9faq.html"> <B>OS-9
FAQ</B></A>:  the OS-9 real-time operating system is what we use in the vehicle.
          Also pertaining to OS-9 is
          <A HREF = "http://iglou.com/~stevenw/windsorhome.html"> Windsor Systems
Home Page</A>.  <P>

          <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/healey.rgb.Z"> Tony
Healey</A> (healey@lex.me.nps.navy.mil),
          <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/mcghee.rgb.Z">  Bob
McGhee</A> (mcghee@cs.nps.navy.mil), and

          <A HREF = "ftp://taurus.cs.nps.navy.mil/pub/auv/brutzman.rgb.Z"> Don
Brutzman</A> (brutzman@nps.navy.mil)
          <A HREF = "http://www.cs.indiana.edu/finger/gateway?brutzman@nps.navy.mil">
contact info</A> <P>

      return to <A HREF="ftp://taurus.cs.nps.navy.mil/pub/mosaic/nps_mosaic.html#TOC">
NPS Home Page table of contents</A>

</ul>
```

```
______________________________________________________________________________<P>
The Universal Resource Locator (URL) for this home page is
<B> ftp://taurus.cs.nps.navy.mil/pub/auv/auv.html </B> <P>

<ADDRESS> NPS AUV World-Wide-Web Home Page: <A HREF =
"ftp://taurus.cs.nps.navy.mil/pub/auv/brutzman.rgb.Z"> Don Brutzman</A>
                                        (brutzman@nps.navy.mil)
                                        <A HREF =
"http://www.cs.indiana.edu/finger/gateway?brutzman@nps.navy.mil"> contact info</A> (18
October 94)
</ADDRESS>
```

E. *.sd.tcl* Multicast Backbone (MBone) Configuration File

```
# $Header: sd_start.tcl,v 1.4 (Van Jacobson LBL)
#
# NOTE:         This file was modified from the default .sd.tcl
#       to support automatic execution of multicast DIS
#       virtual world programs for the NPS AUV and NPSNET,
#       and to automatically spawn mosaic when a URL is
#       present in the session advertisement.
#
#       Please report additions, comments and corrections to
#       Don Brutzman, brutzman@nps.navy.mil
#
#       last modified:   19 OCT 94
#
#       URL of this example file is:
#           ftp://taurus.cs.nps.navy.mil/pub/auv/.sd.tcl.auv-uvw
#
# tcl 'hooks' invoked when sd takes some action on a session.
#
# sd will invoke:
#       start_session   when the user asks to 'open' (start) a session
#       create_session  just after the user creates a new session
#       heard_session   when announcement for a session is first heard
#       delete_session  when the user or a timeout deletes a session
#
# When any of the above are invoked, the global array sd_sess
# contains all the information about the session to be started:
#   sd_sess(name)
#   sd_sess(description)
#   sd_sess(address)
#   sd_sess(ttl)
#   sd_sess(creator)
#   sd_sess(creator_id)
#   sd_sess(source_id)
#   sd_sess(arrival_time)
#   sd_sess(start_time)
#   sd_sess(end_time)
#   sd_sess(attributes)                 (list of session attributes)
#   sd_sess(media)              (list of media names)
#
# For each media name there is an array containing the information
# about that media:
#   sd_$media(port)
#   sd_$media(conf_id)
#   sd_$media(attributes)       (list of media attributes)
#
# Media and session attributes are strings of the form "name" or
# "name:value".
#
# Some global state information is available in array sd_priv:
#   sd_priv(audio)              (= 0 if audio disabled with -a)
#   sd_priv(video)              (= 0 if video disabled with -v)
#   sd_priv(whiteboard)                 (= 0 if wb disabled with -w)

proc start_session {} {
        global sd_sess sd_priv mosaic

        # thanks to Bill Fenner of Navy Research Lab for posting this addition.
        # start up Mosaic if there is a URL in the description.
        if {[regexp {[a-zA-Z]+://[^ ]+} $sd_sess(description) url]} {
                exec $mosaic -home $url &
        }

        # invoke the appropriate start proc for each of the media
```

```
        # if such a proc exists and that media is enabled.
        foreach m $sd_sess(media) {
                if { [llength [info proc start_$m]] && $sd_priv($m) } {
                        start_$m
                }
        }
}

proc start_audio {} {
        global sd_sess sd_audio
        set audiofmt ""
        set packetfmt "-n"
        foreach a $sd_audio(attributes) {
                case $a {
                        fmt:* { set audiofmt [string range $a 4 end] }
                        vt  { set packetfmt "-v" }
                }
        }
        set confaddr [format "%s/%s/%s/%s/%s" $sd_sess(address) \
                $sd_audio(port) $sd_audio(conf_id) $audiofmt $sd_sess(ttl)]

        global vat
        exec $vat -C $sd_sess(name) $packetfmt $confaddr &
}

proc start_video {} {
        global sd_sess sd_video
        set videofmt "nv"
        foreach a $sd_video(attributes) {
                case $a {
                        fmt:* { set videofmt [string range $a 4 end] }
                }
        }
        case $videofmt {
                nv {
                        global nv
                        exec $nv -ttl $sd_sess(ttl) $sd_sess(address) \
                                $sd_video(port) &
                }
                ivs {
                        global ivs
                        exec $ivs -a -r -T $sd_sess(ttl) \
                                dest $sd_sess(address) &
                }
                jpg {
                        global imm
                        exec $imm -p $sd_video(port) -I $sd_sess(address) \
                              -ttl $sd_sess(ttl) -n $sd_sess(name) &
                }

        }
}

proc start_whiteboard {} {
        global sd_sess sd_whiteboard wb
        set orient "-l"
        foreach a $sd_whiteboard(attributes) {
                case $a {
                        orient:portrait { set orient -p }
                        orient:landscape { set orient -l }
                        orient:seascape { set orient +l }
                        orient:dis-npsnet {
                                global nps
                                exec $nps -F "config.trg" -p $sd_video(port) \
                                        -I $sd_sess(address) -t $sd_sess(ttl)
&
                                # do not also execute whiteboard
```

```
                                return
                        }
                        orient:dis-auv-uvw {
                                global  dis_auv_uvw
                                global  dis_auv_uvw_dir
                                cd      $dis_auv_uvw_dir
                                exec    $dis_auv_uvw -port
$sd_whiteboard(port)    \
                                                -address $sd_sess(address) &
                                cd      ..
                                # do not also execute whiteboard
                                return
                        }
                }
        }
        global xpsview
        # running xpsview is a bug fix to prevent irix 5.2 crash
        puts stdout  " "
        puts stdout  "Spawning dummy xpsview to avoid whiteboard bug crash under
Irix 5.2 ..."
        exec $xpsview  &
        exec $wb -t $sd_sess(ttl) -C wb:$sd_sess(name) $orient \
                $sd_sess(address)/$sd_whiteboard(port) &
}

proc create_session {} {
}

proc heard_session {} {
}

proc delete_session {} {
}

##########################################################################
# set up media option menus for new session windows.

set sd_menu(whiteboard)         "orient: portrait landscape seascape\
                        dis-npsnet dis-auv-uvw"

set sd_menu(audio)      "fmt: pcm pcm2 pcm4 idvi dvi2 dvi4 gsm lpc4 nvt"
set sd_menu(video)      "fmt: nv ivs imm"

##########################################################################

# set up the command names

# Edit these to match your system locations & filenames, if necessary.
# The version of mosaic that you point to must not have a precompiled
#   local home page hard-wired in the binary, or it won't redirect.

set vat                         "/n/elsie/work3/macedoni/video/vat"
set nv                  "/n/elsie/work3/macedoni/video/nv"
set ivs                         "/n/elsie/work3/macedoni/video/ivs"
set wb                  "/n/elsie/work3/macedoni/video/wb_src/wb"
set imm                         "/n/elsie/work3/macedoni/video/imm"

set mosaic              "/n/dude/work/brutzman/xmosaic/Mosaic-sgi"
set xpsview             xpsview

set npsnet              "/n/bossie/work3/npsnetIV/npsnetIV"

set dis_auv_uvw_dir     "/n/dude/work/brutzman/auv-uvw"
set dis_auv_uvw                 viewer
```

F. *.mailcap* Configuration File for *Mosaic* Multimedia Viewers

```
###########################################################################
# global mailcap file for mosaic on NPS gravy net   Don Brutzman   18 OCT 94 #
###########################################################################
# global unix system location:          /usr/local/lib/mosaic/mailcap       #
# users can put their own preferences in  ~yourname/.mailcap                #
#                                                                           #
# edit directory paths as necessary for your machine                        #
#                                                                           #
# reference:  http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/mailcap.html #
#                                                                           #
#       URL of this example file is:                                        #
#           ftp://taurus.cs.nps.navy.mil/pub/auv/.mailcap.auv-uvw           #
#                                                                           #
###########################################################################

application/postscript; xpsview %s

###########################################################################

# If you wish to use xplaygizmo ( ftp://ftp.ncsa.uiuc.edu/Mosaic/misc/ )

# video/mpeg; xplaygizmo -p mpeg_play %s

# Alternate mpeg_player which includes looping capability:

video/mpeg; /n/dude/work/brutzman/xmosaic/mpeg_player %s

# I don't think xplaygizmo is needed for small audio, but it is workable.
# It is helpful on longer files such as Internet Talk Radio.

audio/*; /n/dude/work/brutzman/xmosaic/xplaygizmo -p -q sfplay %s

# audio/*; sfplay %s

###########################################################################
#
# STILL NEEDED: dvi viewer (such as the one which works for mosaic on grus)

  application/x-dvi; xdvi %s

#
# Note:  besides Mosaic, mailcap files are used by MIME-capable
#        multimedia mail handlers
###########################################################################

# Final note:  be sure to select /Options _Reload_Config Files from the menu
#              for any changes to be effective in your current session.
```

G. *.cshrc* Excerpts: Login Configurations for MBone and *Mosaic*

```
##################################################################

#   mbone aliases/paths                   [from ~brutzman/.cshrc]

#   sd           to start the session directory tool, everything else
#                is invoked from the sd menu

set path = ($path ~macedoni/video        ~macedoni/bin)
set path = ($path ~macedoni/video/wb_src ~macedoni/video/mm_src)

alias sd      '~macedoni/video/sd &'             # session directory
alias vat     '~macedoni/video/vat'              # visual audio tool
alias nv      '~macedoni/video/nv'               # net video
alias wb      '~macedoni/video/wb_src/wb'        # white board
alias xv      '~macedoni/video/xv'               # xview
alias imm     '~macedoni/video/imm'              # image monitor tool
alias imm2xv '~macedoni/video/mm2xv'             # image monitor => xview
alias ivs     '~macedoni/video/ivs'              # INRIA videoconferencing system
#             normally imm goes full screen, imm can use imm2xv for a window

# also copy the file ~brutzman/.sd.tcl to your home account
# so that the tools are initially set up properly!

# subscribe to local NPS list npsmbone@nps.navy.mil if you are interested

##################################################################

# mosaic/world-wide-web browser aliases/paths - for gravynet users

set path = ($path ~brutzman/xmosaic)

alias mosaic '~brutzman/xmosaic/Mosaic-sgi
ftp://taurus.cs.nps.navy.mil/pub/mosaic/nps_mosaic.html#TOC &'

alias www    '~brutzman/xmosaic/www
ftp://taurus.cs.nps.navy.mil/pub/mosaic/nps_mosaic.html '

alias lynx   '~brutzman/xmosaic/lynx
ftp://taurus.cs.nps.navy.mil/pub/mosaic/nps_mosaic.html '

# also copy the .mailcap file to initialize mosaic

#     cp ~brutzman/.mailcap .mailcap

##################################################################

#       URL of this example file is:
#           ftp://taurus.cs.nps.navy.mil/pub/auv/.cshrc-excerpt.auv-uvw
```

LIST OF REFERENCES

Bailey, Michael P., "Object-Oriented Simulation Pictures (OOSPICs) for Design and Testing," *Proceedings of the 1994 Summer Computer Simulation Conference*, San Diego California, July 18-20 1994, pp. 67-76.

Berners-Lee, Tim, Cailliau, Luotonen, Ari, Nielsen, Henrik Frystyk, and Secret, Arthur, "The World-Wide Web," *Communications of the ACM*, vol. 37 no. 8, August 1994, pp. 76-82.

Berners-Lee, Tim and Frystyck, Henrik, *www* World-Wide Web hypertext browser, *http://info.cern.ch/hypertext/WWW/TheProject.html*, European Particle Physics Laboratory (CERN), Geneva Switzerland, 1994.

Brutzman, Donald P., *NPS AUV Integrated Simulator*, Master's Thesis, Naval Postgraduate School, Monterey California, March 1992.

Brutzman, Donald P., Kanayama, Yutaka, and Zyda, Michael J., "Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles," *Proceedings of the IEEE Oceanic Engineering Society Conference AUV 92*, Washington DC, June 2-3 1992, pp. 3-10.

Brutzman, Donald P., "From virtual world to reality: designing an autonomous underwater robot," *AAAI Fall Symposium on Applications of Artificial Intelligence to Real-World Autonomous Mobile Robots*, Cambridge Massachusetts, October 23-25 1992, pp. 18-22.

Brutzman, Donald P., *A Virtual World for an Autonomous Underwater Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey California, December 1994. The accompanying public electronic distribution of this reference includes source code and executable programs. World-Wide Web (WWW) Uniform Resource Locator (URL) is *ftp://taurus.cs.nps.navy.mil/pub/auv/auv.html*

Cooke, J.C., Zyda, M.J., Pratt, D.R. and McGhee, R.B., "NPSNET: Flight Simulation Dynamic Modeling using Quaternions," *PRESENCE: Teleoperations and Virtual Environments*, vol. 1 no. 4, Fall 1992, pp. 404-420.

Fu, K.S., Gonzalez, R.C. and Lee, C.S.G., *Robotics: Control, Sensing, Vision and Intelligence*, McGraw-Hill, New York, 1987.

Foley, James D, van Dam, Andries, Feiner, Steven K. and Hughes, John F., *Computer Graphics: Principles and Practice*, second edition, Addison-Wesley, Reading Massachusetts, 1990.

Hughes, Kevin, "Entering the World-Wide Web: A Guide to Cyberspace," *ftp://ftp.eit.com/pub/web.guide/guide.61/guide.61.ps.Z*, May 1994.

IEEE Standard for Information Technology - Protocols for Distributed Interactive Simulation (DIS) Applications, version 2.0, third draft, Institute for Simulation and Training report IST-CR-93-15, University of Central Florida, Orlando Florida, May 28 1993.

Kuroda, Yoji, Ura, Tamaki, and Aramaki, Koji, "Vehicle Control Architecture for Operating Multiple Vehicles," *Proceedings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, IEEE Oceanic Engineering Society, Cambridge Massachusetts, July 19-20 1994, pp. 323-329.

Macedonia, Michael R. and Brutzman, Donald P., "MBone Provides Audio and Video Across the Internet," *COMPUTER*, pp. 30-36, April 1994.

Marco, David B., *Slow Speed Control and Dynamic Positioning of an Autonomous Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey California, March 1995.

Microware Systems Corporation, *Using Professional OS-9*, version 2.4, Des Moines Iowa, 1991.

Microware Systems Corporation, *OS-9 C Compiler User's Manual*, version 3.1, Des Moines Iowa, 1991.

Montulli, Lou et al., *lynx*, World-Wide Web hypertext browser, *http://kufacts.cc.ukans.edu/about_lynx/about_lynx.html*, University of Kansas, 1994.

National Center for Supercomputer Applications, *Mosaic*, World-Wide Web hypertext browser, *http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html*, University of Illinois, Urbana-Champaign Illinois, 1994.

National Center for Supercomputer Applications, "A Beginner's Guide to HTML," *http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html*, University of Illinois, Urbana-Champaign Illinois, 1994.

Osterhout, John K., *Tcl and the Tk Toolkit*, Addison-Wesley, Reading Massachusetts, 1994.

Trimble, G. M., "Effecting Heuristic Constraint and Adaptive Mission Execution in a Directed-Task UUV Control Architecture," *International Advanced Robotics Programme: Mobile Robots for Subsea Environments*, Monterey Bay Aquarium Research Institute (MBARI), Monterey California, May 3-6, 1994.

Welch, Brent, *Practical Programming in Tcl and Tk*, Prentice Hall, to appear.

Wernecke, Josie, Open Inventor Architecture Group, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*, Addison-Wesley, Reading Massachusetts, 1994.

Wernecke, Josie, Open Inventor Architecture Group, *The Inventor Toolmaker: Extending Open Inventor, Release 2*, Addison-Wesley, Reading Massachusetts, 1994.

Williams, Thomas and Kelley, Colin, *GNUPLOT: An Interactive Plotting Program*, version 3.5, *"http://www.cs.dartmouth.edu/gnuplot/"*, 1994.

Zeswitz, Steven, *NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange*, Masters Thesis, Naval Postgraduate School, Monterey California, 28 May 1993.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria VA 22304-6145	2
2.	Library, Code 052 Naval Postgraduate School Monterey CA 93943-5101	2
3.	Computer Technology Programs, Code CS Naval Postgraduate School Monterey, California 93943-5000	1
4.	Dr. Michael P. Bailey, Code OR/Ba Operations Research Department Naval Postgraduate School Monterey, California 93943-5000	1
5.	Don Brutzman, Code UW/Br Undersea Warfare Academic Group Naval Postgraduate School Monterey, California 93943-5000	1
6.	Dr. James Eagle, Code UW Chair, Undersea Warfare Academic Group Naval Postgraduate School Monterey, California 93943-5000	1
7.	Dr. Anthony J. Healey, Code ME/Hy Mechanical Engineering Department Naval Postgraduate School Monterey, California 93943-5000	1
8.	Dr. SeHung Kwak, Code CS/Kw Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1

9. Dr. Ted Lewis, Code CS/Lw 1
Chair, Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5000

10. Dr. David Marco, Code ME/Ma 1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, California 93943-5000

11. Dr. Robert B. McGhee, Code CS/Mz 1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5000

12. Dr. Michael J. Zyda, Code CS/Zk 1
Computer Science Department
Naval Postgraduate School
Monterey, California 93943-5000

13. Commander, Naval Undersea Warfare Center Division 1
1176 Howell Street
Attn: Erik Chaum, Code 2251, Building 1171-3
Combat Systems Engineering and Analysis Laboratory (CSEAL)
Newport, Rhode Island 02841-1708

14. Dr. James Bellingham 1
Underwater Vehicles Laboratory, MIT Sea Grant College Program
292 Main Street
Massachusetts Institute of Technology
Cambridge Massachusetts 02142

15. D. Richard Blidberg, Director 1
Marine Systems Engineering Laboratory, Marine Science Center
Northeastern University
East Point, Nahant, Massachusetts 01908

16. Professor Thor I. Fossen 1
University of Trondheim
The Norwegian Institute of Technology
Department of Engineering Cybernetics
N-7034 Trondheim, Norway

17. Clinton N. Keith 1
Applied Remote Technology
9950 Scripps Lake Drive, Suite 106
San Diego California 92131

18. Dr. Yoji Kuroda 1
Institute of Industrial Science
University of Tokyo
7-22-1, Roppongi, Minato-ku, Tokyo 106 Japan

19. Michael Lee 1
Senior Research Engineer
160 Central Avenue
Monterey Bay Aquarium Research Institute (MBARI)
Pacific Grove, California 93950

20. Chris O'Donnel, Disposal Technology Division 1
Commander, Naval Explosive Ordnance Disposal
Indian Head, Maryland 20640-5070

21. Dr. W. Kenneth Stewart 1
Deep Submergence Laboratory
Woods Hole Oceanographic Institution (WHOI)
Woods Hole, Massachusetts 02543

22. Dr. Glen H. Wheless 1
Center for Coastal Physical Oceanography
Crittendon Hall, Old Dominion University
Norfolk Virginia 23529

23. Dr. Dana Yoerger 1
Deep Submergence Laboratory
Woods Hole Oceanographic Institution (WHOI)
Woods Hole, Massachusetts 02543

24. Dr. Junku Yuh 1
Autonomous Systems Laboratory, Department of Mechanical Engineering
University of Hawaii at Manoa
Honolulu Hawaii 96822

25. Dr. David Zeltzer 1
Sensory Communication Group
Research Laboratory of Electronics
Massachusetts Institute of Technology
50 Vassar Street, Room 36-763
Cambridge Massachusetts 02139